Building Security In Maturity Model

BSIMM-V

October 2013

Release 5.0

Authors

BSIMM4 and BSIMM-V: Gary McGraw, Ph.D., Sammy Migues, & Jacob West

BSIMM1 through 3: Gary McGraw, Ph.D., Brian Chess, Ph.D., & Sammy Migues

Acknowledgements

Thanks to the sixty-seven executives from the world-class software security initiatives we studied from around the world to create BSIMM-V. They include: Adobe (Brad Arkin), Aetna (Jim Routh), Bank of America (Jim Apple), Box, Capital One (Keith Gordon), Comerica Bank (George Smirnoff), EMC (Eric Baize), Epsilon (Chris Ray), F-Secure (Antti Vähä-Sipilä), Fannie Mae (Stan Wisseman), Fidelity (David Smith), Goldman Sachs (Phil Venables), HSBC (Malcolm Kelly and Simon Hales), Intel, Intuit, JPMorgan Chase & Co. (Jeff Cohen), Lender Processing Services Inc., Marks and Spencer (Noel Dunne), Mashery (Chris Lippi), McAfee (James Ransome), McKesson (Mike Wilson), Microsoft (Steve Lipner), NetSuite (Brian Chess), Neustar (Jonathan Coombes), Nokia (Janne Uusilehto), Nokia Siemens Networks (Konstantin Shemyak), PayPal (Erick Lee), Pearson Learning Technologies (Aaron Weaver), QUALCOMM (Alex Gantman), Rackspace (Jim Freeman), Salesforce (Robert Fly), Sallie Mae (Jerry Archer), SAP (Gerhard Oswald), Sony Mobile (Per-Olof Persson), Standard Life (Alan Stevens), SWIFT (Peter De Gersem and Alain Desausoi), Symantec (Gary Phillips), Telecom Italia (Marco Bavazzano), Thomson Reuters (Timothy Mathias), TomTom (Xander Heemskerk), Vanguard (Samuel M. D'Amore, Jr.), Visa (Gary Warzala), VMware (Iain Mulholland), Wells Fargo (Steve Adegbite), and Zynga. To those who can't be named, you know who you are, and we could not have done this without you.

Thanks to Gabriele Giuseppini, David Harper, John Holland, Paco Hope, Matias Madou, and Florence Mottay who helped with data collection in Europe. Thanks to Andres Cools, Partha Dutta, Nabil Hannan, Jason Hills, Girish Janardhanudu, Troy Jones, Drew Kilbourne, Todd Lukens, Brian Mizelle, Kabir Mulchandani, Jason Rouse, Joel Scambray, Jay Schulman, Carl Schwarcz, Rajiv Sinha, Mike Ware, Caroline Wong, and Dave Wong for help with U.S. data collection. Thanks to Matteo Meucci (Minded Security), Markus Schumacher (Virtual Forge), and Susana Romaniz and team (UTN-FRSF) and Ivan Arce (Fundación Sadosky) for the translations into Italian, German, and Spanish, respectively. Thanks to Betsy Nichols (PlexLogic) for hard-core statistical analysis in BSIMM2.

Thanks to Pravir Chandra who built a draft maturity model under contract to Fortify Software and thereby sparked this project. Thanks to John Steven for building the first software security framework, described in Chapter 10 of *Software Security*. Thanks to John Steven, Roger Thornton, Mike Ware, Jim DelGrosso, and Robert Hines for helping us hammer out the SSF described here.

Data for The Building Security In Maturity Model was captured by Cigital and HP Fortify. BSIMM model translations by VirtualForge (German), Minded Security (Italian), and UTN-FRSF and Fundación Sadosky (Spanish).

BSIMM-V License

Executive Summary

The Building Security In Maturity Model (BSIMM) is the result of a multi-year study of real-world software security initiatives. We present the model as built directly out of data observed in sixty-seven software security initiatives, from firms including: Adobe, Aetna, Bank of America, Box, Capital One, Comerica Bank, EMC, Epsilon, F-Secure, Fannie Mae, Fidelity, Goldman Sachs, HSBC, Intel, Intuit, JPMorgan Chase & Co., Lender Processing Services Inc., Marks and Spencer, Mashery, McAfee, McKesson, Microsoft, NetSuite, Neustar, Nokia, Nokia Siemens Networks, PayPal, Pearson Learning Technologies, QUALCOMM, Rackspace, Salesforce, Sallie Mae, SAP, Sony Mobile, Standard Life, SWIFT, Symantec, Telecom Italia, Thomson Reuters, TomTom, Vanguard, Visa, VMware, Wells Fargo, and Zynga.

The BSIMM is a measuring stick for software security. The best way to use the BSIMM is to compare and contrast your own initiative with the data about what other organizations are doing contained in the model. You can then identify goals and objectives of your own and look to the BSIMM to determine which additional activities make sense for you.

The BSIMM data show that high maturity initiatives are well rounded—carrying out numerous activities in all twelve of the practices described by the model. The model also describes how mature software security initiatives evolve, change, and improve over time.


VIII
VII
VI
NE
ENE
E

Table of Contents

The Building Security In Maturity Model (BSIMM, pronounced "bee simm") is a study of existing software security initiatives. By quantifying the practices of many different organizations, we can describe the common ground shared by many as well as the variation that makes each unique. Our aim is to help the wider software security community plan, carry out, and measure initiatives of their own. The BSIMM is not a "how to" guide, nor is it a one-size-fits-all prescription. Instead, the BSIMM is a reflection of the software security state of the art.

We begin with a brief description of the function and importance of a software security initiative. We then explain our model and the method we use for quantifying the state of an initiative. Since the BSIMM study began in 2008, we have studied 72 initiatives, which comprise 166 distinct measurements because some firms use the BSIMM to measure each of their business units and some firms have been measured more than once. To ensure the continued relevance of the data we report on, we excluded measurements older than 48 months from BSIMM-V. The current data set comprises 161 distinct measurements collected from 67 firms. Thanks to repeat measurements, we report not only on current practices, but also on the ways in which some initiatives have evolved over a period of years. We devote the later portion of the document to a detailed explanation of the 112 activities that now comprise our model and a summary of the raw data we have collected. We have reviewed the description of each activity for BSIMM-V and also added one new activity to the model.

Our work with the BSIMM model shows that measuring a firm's software security initiative is both possible and extremely useful. BSIMM measurements can be used to plan, structure, and execute the evolution of a software security initiative. Over time, firms participating in the BSIMM project show measurable improvement in their software security initiatives.

Introduction

Software security began to flourish as a discipline separate from computer and network security in the late 1990s. Researchers began to put more emphasis on studying the ways a programmer can contribute to or unintentionally undermine the security of a computer system. What kinds of bugs and flaws lead to security problems? How can we identify problems systematically?

By the middle of the following decade, there was an emerging consensus that creating secure software required more than just smart individuals toiling away. Getting security right means being involved in the software development process.

Since then, practitioners have come to know that process alone is also insufficient. Software security encompasses business, social, and organizational aspects as well. We use the term *Software Security Initiative* to refer to all of the activities undertaken for the purpose of building secure software.

BSIMM-V

The purpose of the BSIMM is to quantify the activities carried out by real software security initiatives. Because these initiatives make use of different methodologies and different terminology, the BSIMM requires a framework that allows us to describe all of the initiatives in a uniform way. Our Software Security Framework (SSF) and activity descriptions provide a common vocabulary for explaining the salient elements of a software security initiative, thereby allowing us to compare initiatives that use different terms, operate at different scales, exist in different vertical markets, or create different work products.

We classify our work as a maturity model because improving software security almost always means changing the way an organization works—something that doesn't happen overnight. We understand that not all organizations need to achieve the same security goals, but we believe all organizations can benefit from using the same measuring stick.

BSIMM-V is the fifth major version of the BSIMM model. It includes updated activity descriptions, one new activity, data from 67 firms, and a longitudinal study.

Audience

The BSIMM is meant for use by anyone responsible for creating and executing a software security initiative. We have observed that successful software security initiatives are typically run by a senior executive who reports to the highest levels in an organization. These executives lead an internal group that we call the Software Security Group (SSG), charged with directly executing or facilitating the activities described in the BSIMM. The BSIMM is written with the SSG and SSG leadership in mind.

We expect readers to be familiar with the software security literature. You can become familiar with many concepts by reading both *Software Security* and *The Security Development Lifecycle.* The BSIMM does not attempt to explain software security basics, describe its history, or provide references to the ever-expanding literature. Succeeding with the BSIMM without becoming familiar with the literature is unlikely.

Method

We built the first version of BSIMM in Fall of 2008 as follows:

- We relied on our own knowledge of software security practices to create the Software Security Framework. (We present the framework on page 19.)
- We conducted a series of nine in-person interviews with executives in charge of software security initiatives. From these interviews, we identified a set of common activities, which we organized according to the Software Security Framework.
- We then created scorecards for each of the nine initiatives that show which activities the initiatives carry out. In order to validate our work, we asked each participant to review the framework, the practices, and the scorecard we created for their initiative.

At this point in the project, we have used the same interview technique to conduct BSIMM assessments for 63 additional firms (a total of 72 firms). (In nine cases, we computed the score for a large firm by scoring some number

of major divisions and then combining the scores into one score for the firm.) Beginning with BSIMM-V, we introduce a data freshness requirement to exclude measurements older than 48 months. This requirement caused five firms to be removed from BSIMM-V (Aon, The Depository Trust & Clearing Corporation (DTCC), Google, Scripps Networks, and an anonymous participant), resulting in a data set of 161 distinct measurements collected from 67 firms. As the data set ages, we intend to decrease the freshness window to 36 months to better align with business cycles.

We used the resulting scores to refine the set of activities and their placement in the framework. We have also conducted a second complete set of interviews with twenty-one of the participants in order to study how their initiatives have changed over time. Four participants have undertaken three such measurements.

We hold the scorecards for individual firms in confidence, but we publish aggregate data describing the number of times we have observed each activity (see page 60). We also publish observations about subsets (such as industry verticals) when our sample size for the subset is large enough to guarantee the anonymity of the participants.

As a descriptive model, the goal of the BSIMM is only to observe and report. We like to say that we wandered off into the jungle to see what we could see and discovered that "monkeys eat bananas in X of the Y jungles we visited." Notice that the BSIMM does not report "you should only eat yellow bananas," "do not run while eating a banana," "thou shalt not steal thy neighbors' bananas," or any other value judgments. Simple observations, simply reported.

Our "just the facts" approach is hardly novel in science and engineering, but in the realm of software security it has not previously been applied at this scale. Previous work has either described the experience of a single organization or offered prescriptive guidance based on a combination of personal experience and opinion.

Participants

The 67 participating organizations are drawn from twelve verticals (with some overlap): financial services (26), independent software vendors (25), cloud (16), technology firms (14), telecommunications (5), retail (4), security (4), healthcare (3), media (3), insurance (2), energy (1), and internet service provider (1). Those companies among the 67 who graciously agreed to be identified include: Adobe, Aetna, Bank of America, Box, Capital One, Comerica Bank, EMC, Epsilon, F-Secure, Fannie Mae, Fidelity, Goldman Sachs, HSBC, Intel, Intuit, JPMorgan Chase & Co., Lender Processing Services Inc., Marks and Spencer, Mashery, McAfee, McKesson, Microsoft, NetSuite, Neustar, Nokia, Nokia Siemens Networks, PayPal, Pearson Learning Technologies, QUALCOMM, Rackspace, Salesforce, Sallie Mae, SAP, Sony Mobile, Standard Life, SWIFT, Symantec, Telecom Italia, Thomson Reuters, TomTom, Vanguard, Visa, VMware, Wells Fargo, and Zynga.

On average, the 67 participants had practiced software security for about 4.25 years at the time of assessment (with some initiatives being brand new at first measurement and the oldest initiative being eighteen years old in October 2013). All 67 firms agree that the success of their program hinges on having an internal group devoted to software security—the SSG. SSG size on average is 14.78 people (smallest 1, largest 100, median 7) with a "satellite" of others (developers, architects, and people in the organization directly engaged in and promoting software security) of 29.6 people (smallest 0, largest 400, median 4). The average number of developers among our targets was 4,190 people (smallest 11, largest 30,000, median 1,600), yielding an average percentage of SSG to development of 1.4%.

All told, the BSIMM describes the work of 975 SSG members working with a satellite of 1,953 people to secure the software developed by 272,358 developers.

Objectives

We created the BSIMM in order to learn how software security initiatives work and to provide a resource for people looking to create or improve their own software security initiative.

In general, any software security initiative will have been created with some high-level goals in mind. The BSIMM is appropriate if your business goals for software security include:

- Informed risk management decisions
- Clarity on what is "the right thing to do" for everyone involved in software security
- Cost reduction through standard, repeatable processes
- Improved code quality

By clearly noting objectives and by tracking practices with metrics tailored to your own initiative, you can use the BSIMM as a measurement tool to guide your own software security initiative.

Instilling software security into an organization takes careful planning and always involves broad organizational change. By using the BSIMM as a guide for your own software security initiative, you can leverage the many years of experience captured in the model. You should tailor the activities that the BSIMM describes to your own organization (carefully considering the objectives we document). Note that no organization carries out all of the activities described in the BSIMM.

Terminology

Nomenclature has always been a problem in computer security, and software security is no exception. There are a number of terms we use in the BSIMM that have particular meanings for us. Here are some of the most important terms we use throughout the document:

Activity – Actions carried out or facilitated by the SSG as part of a practice. Activities are divided into three maturity levels in the BSIMM. Each activity is directly associated with an objective.
Domain – One of the four major groupings in the Software Security Framework. The domains are: governance, intelligence, SSDL touchpoints, and deployment. See the SSF section below.
Practice – One of the twelve categories of BSIMM activities. Each domain in the Software Security Framework has three practices. Activities in each practice are divided into three levels corresponding to maturity. See the SSF section below.
Satellite – A group of interested and engaged developers, architects, software managers, and testers who have a natural affinity for software security and are organized and leveraged by a software security initiative.
Secure Software Development Lifecycle (SSDL) – Any SDLC with integrated software security checkpoints and activities.
Security Development Lifecycle (SDL) – A term used by Microsoft to describe their Secure Software Development Lifecycle.
Software Security Framework (SSF) – The basic structure underlying the BSIMM, comprising twelve practices divided into four domains. See the SSF section below.
Software Security Group (SSG) – The internal group charged with carrying out and facilitating software security. We've observed that step one of a software security initiative is forming an SSG.
Software Security Initiative – An organization-wide program to instill, measure, manage, and evolve software security activities in a coordinated fashion. Also known in the literature as an Enterprise Software Security Program (see chapter 10 of *Software Security*).

Roles

Determining who is supposed to carry out the activities described in the BSIMM is an important part of making any software security initiative work.

Executive Leadership

Of primary interest is identifying and empowering a senior executive to manage operations, garner resources, and provide political cover for a software security initiative. Grassroots approaches to software security sparked and led solely by developers and their direct managers have a poor track record in the real world. Likewise, initiatives spearheaded by resources from an existing network security group often run into serious trouble when it comes time to interface with development groups. By identifying a senior executive and putting him or her in charge of software security directly, you address two management 101 concerns—accountability and empowerment. You also create a place in the organization where software security can take root and begin to thrive.

The executives in charge of the software security initiatives we studied have a variety of titles, including: Director of IT Security and Risk Management, Chief Information Risk Officer, Director of Application Controls, VP Product and Operations, Product Security Manager, Sr. Manager of Product Security, SVP of Global Risk Management, Global Head of Information Security Markets, SVP of Information Security, and CISO. We also observed a fairly wide spread in exactly where the SSG is situated in the firms we studied. In particular, 19 exist in the CIO's organization, 15 exist in the CTO's organization, 14 report to the COO, 4 exist in either the General Counsel's office or the Office of Compliance and Risk Management, 3 SSGs report to CSOs, and one reports directly to the founder or CEO. A number of the companies we studied did not specify where their SSG fits in the larger organization.

The Software Security Group (SSG)

The second most important role in a software security initiative after the senior executive is that of the Software Security Group. Every single one of the 67 programs we describe in the BSIMM has an SSG. Carrying out the activities in the BSIMM successfully without an SSG is very unlikely (and has never been observed in the field to date), so create an SSG before you start working to adopt the BSIMM activities. The best SSG members are software security people, but software security people are often impossible to find. If you must create software security types from scratch, start with developers and teach them about security. Do not attempt to start with network security people and teach them about software, compilers, SDLCs, bug tracking, and everything else in the software universe. No amount of traditional security knowledge can overcome software cluelessness.

SSGs come in a variety of shapes and sizes. All good SSGs appear to include both people with deep coding experience and people with architectural chops. As you will see below, software security can't only be about finding specific bugs such as the OWASP Top Ten. Code review is a very important best practice, and to perform code review you must actually understand code (not to mention the huge piles of security bugs). However, the best code reviewers sometimes make very poor software architects, and asking them to perform an Architecture Risk Analysis will only result in blank stares. Make sure you cover architectural capabilities in your SSG as well as you do code. Finally, the SSG is often asked to mentor, train, and work directly with hundreds of developers. Communications skills, teaching capability, and good consulting horse sense are must–haves for at least a portion of the SSG staff. For more about this issue, see our informIT article, *You Really Need an SSG* <http://bit.ly/7dqCn8>.

Though no two of the 67 firms we examined had exactly the same SSG structure (suggesting that there is no one set way to structure an SSG), we did observe some commonalities that are worth mentioning. At the highest level of organization, SSGs come in three major flavors: those organized according to technical SDLC duties, those organized by operational duties, and those organized according to internal business units. Some SSGs are highly distributed across a firm, and others are very centralized and policy-oriented. If we look across all of the SSGs in our study, there are several common "subgroups" that are often observed: people dedicated to policy, strategy, and metrics; internal "services" groups that (often separately) cover tools, penetration testing, and middleware development plus shepherding; incident response groups; groups responsible for training development and delivery; externally-facing marketing and communications groups; and, vendor-control groups.

In the statistics reported above, we noted an average ratio of SSG to development of 1.4% across the entire group of 67 organizations we studied. That means one SSG member for every 71 developers when we average the ratios for each participant. The largest SSG was 27.27% and the smallest was 0.03%. To remind you of the particulars in terms of actual bodies, SSG size on average among the 67 firms was 14.7 people (smallest 1, largest 100, median 7).

Everybody Else

Our survey participants have engaged everyone involved in the software development lifecycle as a means of addressing software security.

- **Builders,** including developers, architects, and their managers must practice security engineering, ensuring that the systems that we build are defensible and not riddled with holes. The SSG will interact directly with builders when they carry out the activities described in the BSIMM. Generally speaking, as an organization matures, the SSG attempts to empower builders so that they can carry out most of the BSIMM activities themselves with the SSG helping in special cases and providing oversight. In this version of the BSIMM, we often don't explicitly point out whether a given activity is to be carried out by the SSG or by developers or by testers, although in some cases we do attempt to clarify responsibilities in the goals associated with activity levels within practices. You should come up with an approach that makes sense for your organization and takes into account workload and your software lifecycle.
- **Testers** concerned with routine testing and verification should do what they can to keep a weather eye out for security problems. Some of the BSIMM activities in the *Security Testing* practice can be carried out directly by QA.
- **Operations** people must continue to design reasonable networks, defend them, and keep them up. As you will see in the *Deployment* domain of the SSF, software security does not end when software is "shipped."
- **Administrators** must understand the distributed nature of modern systems and begin to practice the principle of least privilege, especially when it comes to applications they host or attach to as services in the cloud.
- **Executives and middle management**, including Line of Business owners and Product Managers, must understand how early investment in security design and security analysis affects the degree to which users will trust their products. Business requirements should explicitly address security needs. Any sizeable business today depends on software to work. Software security is a business necessity.
- **Vendors,** including those who supply COTS, custom software, and software-as-a-service, are increasingly subjected to SLAs and reviews (such as vBSIMM) that help ensure products are the result of a secure SDLC.

Participants feel the most important people to enlist for near-term progress in software security are the builders. Only by pushing past the standard-issue operations view of security will we begin to make software systems that can stand up under attack.

Identifying a Satellite

In addition to the SSG, many software security programs have identified a number of individuals (often developers, testers, and architects) who share a basic interest in software security, but are not directly employed in the SSG. We call this group the *satellite*.

Sometimes the satellite is widely distributed, with one or two members in each product group. Sometimes the satellite is more focused and gets together regularly to compare notes, learn new technologies, and expand the understanding of software security in an organization. Identifying and fostering a strong satellite is important to the success of many software security initiatives (but not all of them). Some BSIMM activities target the satellite explicitly.

Of particular interest, all ten firms with the highest BSIMM scores have a satellite (100%), with an average satellite size of 148 people. Outside of the top ten, 29 of the remaining 57 firms have a satellite (51%). Of the ten firms with the lowest BSIMM scores, only 2 have a satellite (20%), with an average size of less than half a person. This suggests that as a software security initiative matures, its activities become distributed and institutionalized into the organizational structure. Among our population of 67 firms, initiatives tend to evolve from centralized and specialized in the beginning to decentralized and distributed (with an SSG at the core orchestrating things).

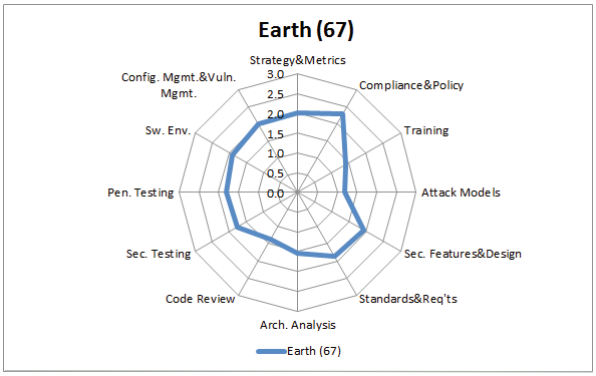
Putting BSIMM-V to Use

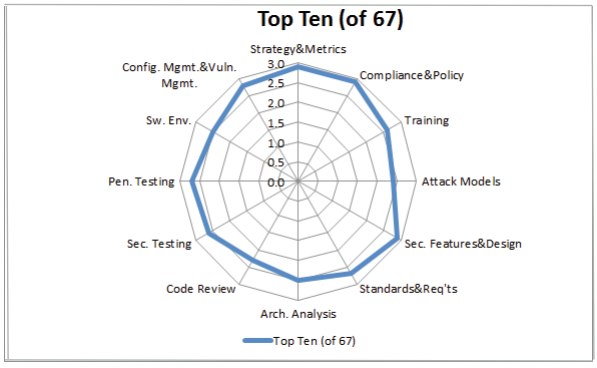
The BSIMM describes 112 activities that any organization can put into practice. The activities are described in terms of the SSF, which identifies twelve practices grouped into four domains. Associated with each activity is an objective.

BSIMM-V Results

The BSIMM data yield very interesting analytical results. Here are two "spider charts" showing average maturity level in each of the twelve practices over some number of organizations. The first graph shows data from all 67 BSIMM-V firms. The second graph shows data from the top ten firms (as determined by raw score computed by summing the activities).

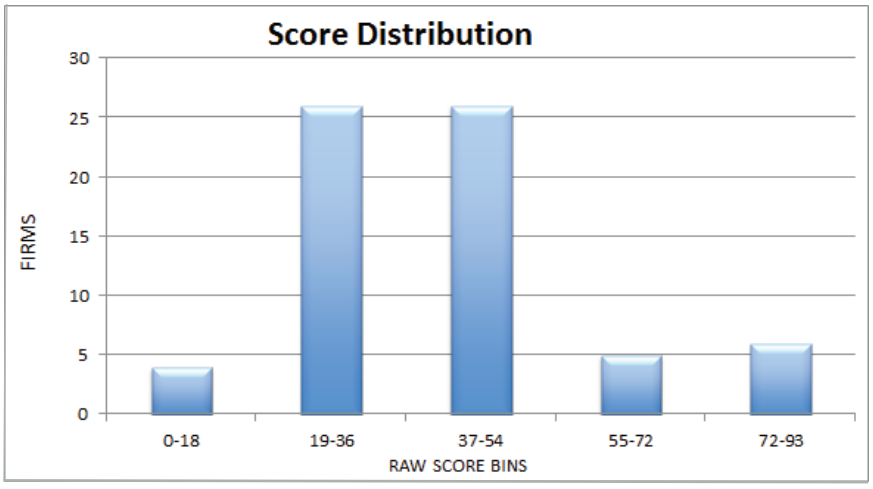
Spider charts are created by noting the highest level of activity in a practice for a given firm (a "high water mark"), and then averaging those scores for a group of firms, resulting in twelve numbers (one for each practice). The spider chart has twelve spokes corresponding to the twelve practices. Note that in all of these charts, level 3 (outside edge) is considered more mature than level 0 (inside point). Other more sophisticated analyses are possible, of course, and we continue to experiment with weightings by level, normalization by number of activities, and other schemes.





By computing a raw score for each firm in the study, we can also take a look at relative maturity and average maturity for one firm against the others. To date, the range of observed scores is [13, 93].

The graph below shows the distribution of scores among the population of 67 participating firms. To make this graph, we divided the scores into five equal bins. As you can see, the scores represent a slightly skewed bell curve.



We are pleased that the BSIMM study continues to grow (the data set has grown just over 75% since publication of BSIMM4 and is over eighteen times the size it was for the original publication). Note that once we exceeded a sample size of thirty firms, we began to apply statistical analysis yielding statistically significant results.

The BSIMM-V Scorecard below shows the number of times each of the 112 activities was observed in the BSIMM-V data. An expanded version of this chart can be found on page 60.

Governance		Intelligence		SSDL Touchpoints		Deployment	
Activity	Observed	Activity	Observed	Activity	Observed	Activity	Observed
[SM1.1]	44	[AM1.1]	21	[AA1.1]	56	[PT1.1]	62
[SM1.2]	34	[AM1.2]	43	[AA1.2]	35	[PT1.2]	51
[SM1.3]	34	[AM1.3]	30	[AA1.3]	24	[PT1.3]	43
[SM1.4]	57	[AM1.4]	12	[AA1.4]	42	[PT2.2]	24
[SM1.6]	36	[AM1.5]	42	[AA2.1]	10	[PT2.3]	27
[SM2.1]	26	[AM1.6]	16	[AA2.2]	8	[PT3.1]	13
[SM2.2]	31	[AM2.1]	7	[AA2.3]	20	[PT3.2]	8
[SM2.3]	27	[AM2.2]	11	[AA3.1]	11		
[SM2.5]	20	[AM3.1]	4	[AA3.2]	4		
[SM3.1]	16	[AM3.2]	6				
[SM3.2]	6						
[CP1.1]	43	[SFD1.1]	54	[CR1.1]	24	[SE1.1]	34
[CP1.2]	52	[SFD1.2]	53	[CR1.2]	34	[SE1.2]	61
[CP1.3]	45	[SFD2.1]	26	[CR1.4]	50	[SE2.2]	31
[CP2.1]	24	[SFD2.2]	29	[CR1.5]	23	[SE2.4]	25
[CP2.2]	28	[SFD3.1]	9	[CR1.6]	25	[SE3.2]	10
[CP2.3]	29	[SFD3.2]	13	[CR2.2]	10	[SE3.3]	9
[CP2.4]	25	[SFD3.3]	9	[CR2.5]	15		
[CP2.5]	35			[CR2.6]	18		
[CP3.1]	14			[CR3.2]	4		
[CP3.2]	11			[CR3.3]	6		
[CP3.3]	8			[CR3.4]	1		
[T1.1]	50	[SR1.1]	48	[ST1.1]	51	[CMVM1.1]	59
[T1.5]	29	[SR1.2]	43	[ST1.3]	55	[CMVM1.2]	59
[T1.6]	23	[SR1.3]	45	[ST2.1]	27	[CMVM2.1]	50
[T1.7]	33	[SR1.4]	27	[ST2.4]	13	[CMVM2.2]	44
[T2.5]	9	[SR2.2]	23	[ST3.1]	11	[CMVM2.3]	30
[T2.6]	13	[SR2.3]	19	[ST3.2]	8	[CMVM3.1]	6
[T2.7]	9	[SR2.4]	19	[ST3.3]	6	[CMVM3.2]	6
[T3.1]	4	[SR2.5]	22	[ST3.4]	5	[CMVM3.3]	2
[T3.2]	4	[SR3.1]	8	[ST3.5]	7		
[T3.3]	8	[SR3.2]	12				
[T3.4]	9						
[T3.5]	5						

New Activities for BSIMM6 Observation

For the second time in the BSIMM project, we have observed an activity not yet captured in the model. We therefore added one new activity to the model going forward. The observations related to this activity will be first reported in the data released with BSIMM6, but we describe the activity in this release.

The two new activities first described in BSIMM4 are being actively tracked during measurements; however, there is some lag in the data for these activities corresponding to remeasurement. We do not retroactively give credit for new activities in past measurements. (This is why, for example, [CR3.4 Automate malicious code detection] is reported as a one in BSIMM-V even though we know a larger number of participating firms undertake this activity.)

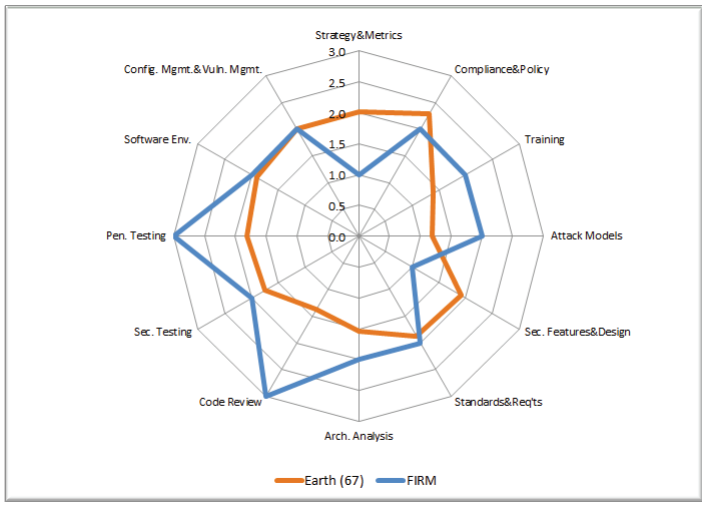
Our criteria for adding an activity to the BSIMM are as follows. If we observe a candidate activity not yet in the model, we determine based on previously captured data and BSIMM mailing list queries how many firms probably carry out that activity. If the answer is multiple firms, we take a closer look at the proposed activity and figure out how it fits with the existing model. If the answer is only one firm, the candidate activity is tabled as too specialized. Furthermore, if the candidate activity is covered by the existing activities or simply refines or bifurcates an existing activity, it is dropped.

Using the criteria above, the activity added to the BSIMM-V model is: [CMVM3.4 Operate a bug bounty program]. We include a complete description of the new activity in the detailed BSIMM-V description and the BSIMM skeleton below.

Measuring Your Firm with BSIMM-V

The most important use of the BSIMM is as a measuring stick to determine where your approach currently stands relative to other firms. Note which activities you already have in place, and use "activity coverage" to determine their levels and build a scorecard. In our own work using the BSIMM to assess levels, we found that the spider-graph-yielding "high water mark" approach (based on the three levels per practice) is sufficient to get a low-resolution feel for maturity, especially when working with data from a particular vertical or geography.

One meaningful comparison is to chart your own maturity high water mark against the averages we have published to see how your initiative stacks up. Below, we have plotted data from a (fake) firm against the BSIMM Earth graph.



BSIMM-V Scorecard for: FIRM — Raw Score: 37

Governance			Intelligence			SSDL Touchpoints			Deployment		
Activity	BSIMM-V Firms	FIRM	Activity	BSIMM-V Firms	FIRM	Activity	BSIMM-V Firms	FIRM	Activity	BSIMM-V Firms	FIRM
[SM1.1]	44	1	[AM1.1]	21	1	[AA1.1]	56	1	[PT1.1]	62	1
[SM1.2]	34		[AM1.2]	43		[AA1.2]	35	1	[PT1.2]	51	1
[SM1.3]	34	1	[AM1.3]	30		[AA1.3]	24	1	[PT1.3]	43	
[SM1.4]	57	1	[AM1.4]	12	1	[AA1.4]	42		[PT2.2]	24	1
[SM1.6]	36		[AM1.5]	42	1	[AA2.1]	10		[PT2.3]	27	
[SM2.1]	26		[AM1.6]	16		[AA2.2]	8	1	[PT3.1]	13	1
[SM2.2]	31		[AM2.1]	7		[AA2.3]	20		[PT3.2]	8	
[SM2.3]	27		[AM2.2]	11	1	[AA3.1]	11				
[SM2.5]	20		[AM3.1]	4		[AA3.2]	4				
[SM3.1]	16		[AM3.2]	6							
[SM3.2]	6										
[CP1.1]	43	1	[SFD1.1]	54		[CR1.1]	24		[SE1.1]	34	
[CP1.2]	52		[SFD1.2]	53	1	[CR1.2]	34	1	[SE1.2]	61	1
[CP1.3]	45	1	[SFD2.1]	26		[CR1.4]	50	1	[SE2.2]	31	1
[CP2.1]	24		[SFD2.2]	29		[CR1.5]	23		[SE2.4]	25	
[CP2.2]	28		[SFD3.1]	9		[CR1.6]	25	1	[SE3.2]	10	
[CP2.3]	29		[SFD3.2]	13		[CR2.2]	10		[SE3.3]	9	
[CP2.4]	25		[SFD3.3]	9		[CR2.5]	15				
[CP2.5]	35	1				[CR2.6]	18				
[CP3.1]	14					[CR3.2]	4	1			
[CP3.2]	11					[CR3.3]	6				
[CP3.3]	8					[CR3.4]	1				
[T1.1]	50	1	[SR1.1]	48	1	[ST1.1]	51	1	[CMVM1.1]	59	1
[T1.5]	29		[SR1.2]	43		[ST1.3]	55	1	[CMVM1.2]	59	
[T1.6]	23	1	[SR1.3]	45	1	[ST2.1]	27	1	[CMVM2.1]	50	1
[T1.7]	33		[SR1.4]	27	1	[ST2.4]	13		[CMVM2.2]	44	
[T2.5]	9		[SR2.2]	23		[ST3.1]	11		[CMVM2.3]	30	
[T2.6]	13	1	[SR2.3]	19		[ST3.2]	8		[CMVM3.1]	6	
[T2.7]	9		[SR2.4]	19		[ST3.3]	6		[CMVM3.2]	6	
[T3.1]	4		[SR2.5]	22	1	[ST3.4]	5		[CMVM3.3]	2	
[T3.2]	4		[SR3.1]	8		[ST3.5]	7		[CMVM3.4]	0	
[T3.3]	8		[SR3.2]	12							
[T3.4]	9										
[T3.5]	5										

Legend:
- Activity: 112 BSIMM-V activities, shown in 4 domains and 12 practices
- BSIMM Firms: count of firms (out of 67) observed performing each activity
- the most common activity within a practice
- a common activity not observed in this assessment
- a common activity observed in this assessment
- a practice where firm's high-water mark score is below the BSIMM-V average

A direct comparison of all 112 activities is perhaps the most obvious use of the BSIMM. This can be accomplished by building a scorecard using the data printed on page 60.

The scorecard you see here depicts a (fake) firm that performs thirty-seven BSIMM activities (1s in the FIRM columns), including eight activities that are the most common in their respective practices (purple boxes). On the other hand, the firm does not perform the most commonly observed activities in the other four practices (red boxes) and should take some time to determine whether these are necessary or useful to its overall software security initiative. The BSIMM-V Firms column shows the number of observations (currently out of 67) for each activity, allowing the firm to understand the general popularity of an activity among the 67 BSIMM participants.

Once you have determined where you stand with activities, you can devise a plan to enhance practices with other activities suggested by the BSIMM. By providing actual measurement data from the field, the BSIMM makes it possible to build a long-term plan for a software security initiative and track progress against that plan. For the record, there is no inherent reason to adopt all activities in every level for each practice. Adopt those activities that make sense for your organization, and ignore those that don't.

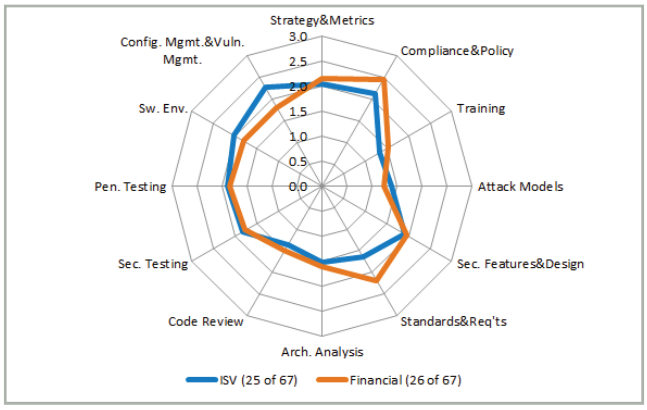
The best way to use the BSIMM in planning is to identify goals and objectives of your own and look to the BSIMM to determine which activities make sense. Choosing between practices is not recommended. In particular, we don't believe a successful initiative will be able to omit all activity for any of the twelve practices. Put another way, the data show that high maturity initiatives are well rounded and carry out activities in all twelve practices. In any case, browsing through objectives and noting which appeal to your culture is a much cleaner way to proceed. Once you know what your objectives are, you can determine which activities to adopt. Please note that in our view it is better to put some of the Level 1 activities in each practice into place than to accelerate to Level 3 in any one practice while ignoring other practices. This view is informed by the data we have gathered.

The breakdown of activities into levels for each practice is meant only as a guide. The levels provide a natural progression through the activities associated with each practice. However, it is not at all necessary to carry out all activities in a given level before moving on to activities at a higher level in the same practice. That said, the levels that we have identified hold water under statistical scrutiny. Level 1 activities (straightforward and simple) are commonly observed, Level 2 (more difficult and requiring more coordination) slightly less so, and Level 3 (rocket science) are much more rarely observed.

By identifying objectives and activities from each practice that would work for you, and by ensuring proper balance with respect to domains, you can create a strategic plan for your software security initiative moving forward. Note that most software security initiatives are multi-year efforts with real budget, mandate, and ownership behind them. Though all initiatives look different and are tailored to fit a particular organization, as we describe below, all initiatives do share common activities.

Studying Groups of Firms in BSIMM-V

The spider charts we introduce above are also useful for comparing groups of firms from particular industry verticals or geographic locations. The graph below shows data from the financial services vertical (26 firms) and independent software vendors (25 firms) charted together. On average, financial services firms have greater maturity as compared to independent software vendors in seven of the twelve practices. By the same measure, independent software vendor have greater maturity as compared to financial services firms in five of the twelve practices. Though there is plenty of overlap here, major differences can be explained with reference to the move to cloud computing among ISVs (see CMVM and SE) while FIs tend to emphasize standards and compliance practices (see SR and CP). A closer look at the 112 activities reveals more interesting differences.



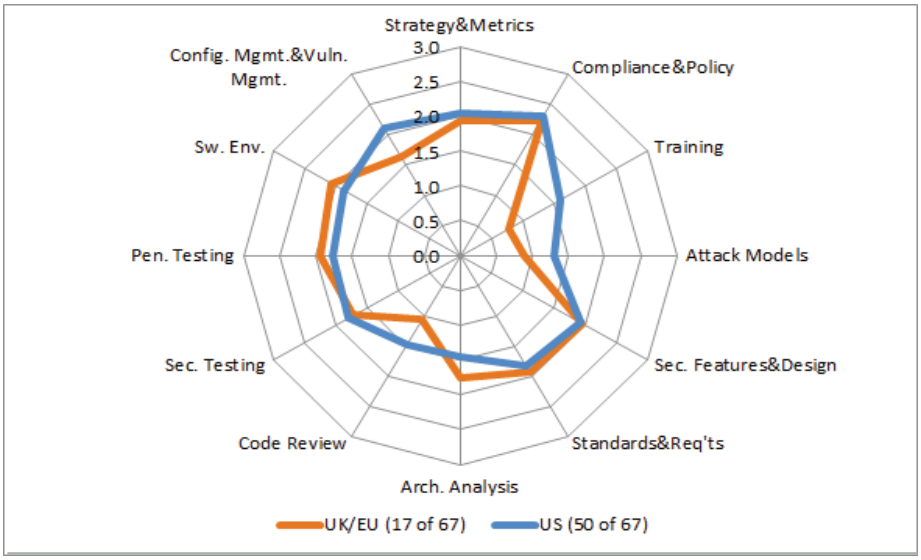
In the table below, you can see the BSIMM Scorecards for the two largest verticals compared side by side. In the Activity columns, we have highlighted in yellow the most common activity in each practice as observed in the entire BSIMM data pool (67 firms). The Delta column shows the absolute value difference between FIs and ISVs for each activity. Large deltas are of particular interest and are highlighted in dark blue when more ISVs have been observed performing the activity and in red when more FIs have been observed performing the activity.

Governance				Intelligence				SSDL Touchpoints				Deployment			
Activity	Financial (of 26)	ISV (of 25)	Delta	Activity	Financial (of 26)	ISV (of 25)	Delta	Activity	Financial (of 26)	ISV (of 25)	Delta	Activity	Financial (of 26)	ISV (of 25)	Delta
[SM1.1]	19	14	5	[AM1.1]	7	9	2	[AA1.1]	23	21	2	[PT1.1]	26	21	5
[SM1.2]	10	17	7	[AM1.2]	21	12	9	[AA1.2]	11	14	3	[PT1.2]	21	19	2
[SM1.3]	16	14	2	[AM1.3]	14	10	4	[AA1.3]	9	11	2	[PT1.3]	16	14	2
[SM1.4]	22	21	1	[AM1.4]	2	6	4	[AA1.4]	21	12	9	[PT2.2]	7	12	5
[SM1.6]	17	12	5	[AM1.5]	19	12	7	[AA2.1]	5	3	2	[PT2.3]	15	9	6
[SM2.1]	11	13	2	[AM1.6]	3	8	5	[AA2.2]	2	3	1	[PT3.1]	4	7	3
[SM2.2]	15	9	6	[AM2.1]	2	6	4	[AA2.3]	7	9	2	[PT3.2]	3	3	0
[SM2.3]	7	13	6	[AM2.2]	4	6	2	[AA3.1]	5	3	2				
[SM2.5]	10	7	3	[AM3.1]	1	3	2	[AA3.2]	0	3	3				
[SM3.1]	8	6	2	[AM3.2]	2	5	3								
[SM3.2]	0	3	3												
[CP1.1]	20	13	7	[SFD1.1]	21	21	0	[CR1.1]	9	9	0	[SE1.1]	15	12	3
[CP1.2]	20	19	1	[SFD1.2]	20	20	0	[CR1.2]	17	12	5	[SE1.2]	25	20	5
[CP1.3]	21	14	7	[SFD2.1]	10	9	1	[CR1.4]	19	19	0	[SE2.2]	12	13	1
[CP2.1]	12	8	4	[SFD2.2]	12	8	4	[CR1.5]	5	14	9	[SE2.4]	6	12	6
[CP2.2]	15	6	9	[SFD3.1]	4	3	1	[CR1.6]	10	9	1	[SE3.2]	2	5	3
[CP2.3]	8	11	3	[SFD3.2]	7	6	1	[CR2.2]	4	5	1	[SE3.3]	3	5	2
[CP2.4]	12	8	4	[SFD3.3]	3	6	3	[CR2.5]	7	4	3				
[CP2.5]	17	11	6					[CR2.6]	10	7	3				
[CP3.1]	12	0	12					[CR3.2]	3	1	2				
[CP3.2]	5	4	1					[CR3.3]	2	3	1				
[CP3.3]	2	5	3					[CR3.4]	1	0	1				
[T1.1]	21	17	4	[SR1.1]	21	17	4	[ST1.1]	22	18	4	[CMVM1.1]	24	22	2
[T1.5]	13	12	1	[SR1.2]	17	18	1	[ST1.3]	23	20	3	[CMVM1.2]	23	24	1
[T1.6]	6	11	5	[SR1.3]	16	16	0	[ST2.1]	11	12	1	[CMVM2.1]	20	20	0
[T1.7]	17	14	3	[SR1.4]	13	9	4	[ST2.4]	6	6	0	[CMVM2.2]	16	21	5
[T2.5]	3	3	0	[SR2.2]	12	7	5	[ST3.1]	3	6	3	[CMVM2.3]	9	14	5
[T2.6]	5	4	1	[SR2.3]	11	5	6	[ST3.2]	0	6	6	[CMVM3.1]	1	4	3
[T2.7]	1	5	4	[SR2.4]	4	9	5	[ST3.3]	1	4	3	[CMVM3.2]	1	4	3
[T3.1]	2	2	0	[SR2.5]	10	4	6	[ST3.4]	1	2	1	[CMVM3.3]	1	1	0
[T3.2]	2	1	1	[SR3.1]	2	5	3	[ST3.5]	2	4	2				
[T3.3]	1	4	3	[SR3.2]	9	2	7								
[T3.4]	5	4	1												
[T3.5]	2	2	0												

There is a great deal of overlap in the most common observations, with a majority of deltas amounting to a count of six or less. In the Compliance and Policy practice, however, there are four activities that are observed significantly more often in FIs than in ISVs. These are [CP1.1 Unify regulatory pressures], [CP1.3 Create policy], [CP 2.2 Require security sign-off for compliance-related risk], and [CP3.1 Create regulatory eye candy]. The fact that FIs put more emphasis on compliance and policy than ISVs do is not surprising given that FIs are regulated. The same goes for data classification (ISVs make things to store data and FIs store lots of data), which explains the sizeable delta in [AM1.2 Create a data classification scheme and inventory]. On the other hand, ISVs tend to rely more on persuasion than coercion when it comes to influencing development teams as reflected by [SM1.2 Create evangelism role and perform internal marketing] and also leverage technology more heavily as seen in [CR1.5 Make code review mandatory for all projects].

When we began working on the BSIMM, we expected the data to lead us toward narratives explaining behavior in particular verticals or geographies. No such luck. For example, though there are small visible differences between the average financial services firm and the average independent software vendor when it comes to software security, the commonalities between high-maturity participants in each vertical outweigh these differences by an order of magnitude.

The same sort of thing can be said regarding European + U.K. firms (seventeen) and U.S. firms (fifty). Differences between sets show European + U.K. initiatives lag U.S. initiatives by twelve to eighteen months, as reflected in the spider chart below, and correspond with the development of the software security market in these regions. For an in depth treatment of the European market, see *BSIMM Europe: Measuring software security initiatives worldwide <http://bit.ly/TTBfX>*.



BSIMM as a Longitudinal Study

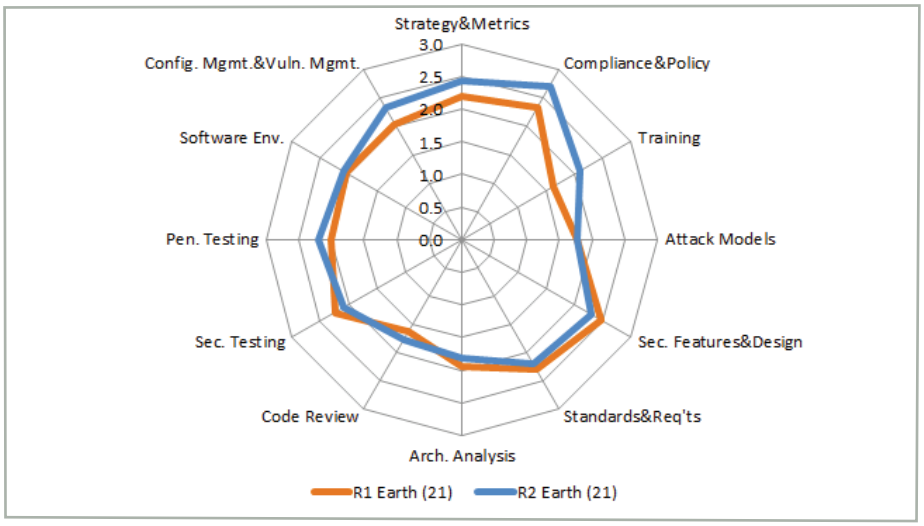
Twenty-one of the 67 firms have been measured twice using the BSIMM measurement system. On average, the time between the two measurements was 24 months. Though individual activities among the twelve practices come and go as shown in the Longitudinal Scorecard below, in general, remeasurement over time shows a clear trend of increased maturity in the population of the twenty-one firms remeasured thus far. The raw score went up in seventeen of the twenty-one firms an average of 14 (an average increase of 16%). Software security initiatives mature over time.

Governance			Intelligence			SSDL Touchpoints			Deployment		
Activity	BSIMM Round 1 (of 21)	BSIMM Round 2 (of 21)	Activity	BSIMM Round 1 (of 21)	BSIMM Round 2 (of 21)	Activity	BSIMM Round 1 (of 21)	BSIMM Round 2 (of 21)	Activity	BSIMM Round 1 (of 21)	BSIMM Round 2 (of 21)
[SM1.1]	13	20	[AM1.1]	8	8	[AA1.1]	17	20	[PT1.1]	20	20
[SM1.2]	15	13	[AM1.2]	14	15	[AA1.2]	15	13	[PT1.2]	14	19
[SM1.3]	14	17	[AM1.3]	11	13	[AA1.3]	15	13	[PT1.3]	13	15
[SM1.4]	18	21	[AM1.4]	10	7	[AA1.4]	12	15	[PT2.2]	8	9
[SM1.6]	15	15	[AM1.5]	13	17	[AA2.1]	8	7	[PT2.3]	8	9
[SM2.1]	11	13	[AM1.6]	7	12	[AA2.2]	3	3	[PT3.1]	6	8
[SM2.2]	10	13	[AM2.1]	9	6	[AA2.3]	9	9	[PT3.2]	5	5
[SM2.3]	13	10	[AM2.2]	7	10	[AA3.1]	5	6			
[SM2.5]	8	12	[AM3.1]	2	3	[AA3.2]	2	2			
[SM3.1]	5	9	[AM3.2]	2	5						
[SM3.2]	3	4									
[CP1.1]	15	15	[SFD1.1]	19	16	[CR1.1]	5	12	[SE1.1]	8	9
[CP1.2]	17	18	[SFD1.2]	12	18	[CR1.2]	12	16	[SE1.2]	19	18
[CP1.3]	19	19	[SFD2.1]	12	10	[CR1.4]	16	20	[SE2.2]	10	11
[CP2.1]	8	6	[SFD2.2]	10	13	[CR1.5]	9	10	[SE2.4]	8	12
[CP2.2]	13	11	[SFD3.1]	6	6	[CR1.6]	11	15	[SE3.2]	4	6
[CP2.3]	11	9	[SFD3.2]	6	5	[CR2.2]	9	7	[SE3.3]	5	2
[CP2.4]	7	13	[SFD3.3]	6	4	[CR2.5]	9	10			
[CP2.5]	13	16				[CR2.6]	5	12			
[CP3.1]	4	8				[CR3.2]	1	3			
[CP3.2]	4	6				[CR3.3]	2	3			
[CP3.3]	4	8				[CR3.4]	0	0			
[T1.1]	16	21	[SR1.1]	16	17	[ST1.1]	15	16	[CMVM1.1]	17	21
[T1.5]	10	16	[SR1.2]	12	19	[ST1.3]	11	15	[CMVM1.2]	17	20
[T1.6]	10	13	[SR1.3]	10	18	[ST2.1]	14	13	[CMVM2.1]	19	19
[T1.7]	10	18	[SR1.4]	9	11	[ST2.4]	6	7	[CMVM2.2]	13	19
[T2.5]	5	5	[SR2.2]	10	12	[ST3.1]	6	6	[CMVM2.3]	10	10
[T2.6]	4	7	[SR2.3]	7	8	[ST3.2]	9	7	[CMVM3.1]	3	3
[T2.7]	8	6	[SR2.4]	10	9	[ST3.3]	4	5	[CMVM3.2]	4	5
[T3.1]	3	4	[SR2.5]	7	12	[ST3.4]	3	3	[CMVM3.3]	0	0
[T3.2]	2	2	[SR3.1]	6	4	[ST3.5]	5	6			
[T3.3]	3	4	[SR3.2]	6	5						
[T3.4]	2	8									
[T3.5]	4	5									

Here are two ways of thinking about the change represented by the longitudinal scorecard. We see the biggest changes in: [T1.7 Deliver on-demand individual training], with 10 new observations; [SM2.1 Publish data about software security internally] and [SR1.3 Translate compliance constraints to requirements], with nine new observations; and [SM2.5 Identify metrics and use them to drive budgets], [CP2.4 Paper all vendor contracts with software security SLAs], and [CR1.1 Create a top N bugs list (real data preferred)], with eight new observations. There are an additional five activities newly observed in seven remeasured firms.

Less obvious from the scorecard is the "churn" among activities. For example, while the count of firms remained constant for [CMVM2.3 Develop an operations inventory of applications], five firms have new observations of this activity, while the activity was no longer observed in five other firms. Similarly, we had new observations in five firms for [T3.5 Establish SSG office hours] and [SR2.3 Create standards for technology stack], while the activity was no longer observed in four firms. In addition, we had new observations in four firms for [SFD3.2 Require use of approved security features and frameworks], while the activity was no longer observed in five firms.

Using the spider diagram, we can plot the high watermarks of the first measurement of the twenty-one firms against their second measurement.



BSIMM Over Time

This is the fifth major release of the BSIMM project. The original study included nine firms and nine distinct measurements. BSIMM2 included 30 firms and 42 distinct measurements (some firms include very large subsidiaries that were independently measured). BSIMM3 included 42 firms, eleven that had been remeasured, for a total set of 81 distinct measurements. BSIMM4 included 51 firms, thirteen of which had been remeasured (with one firm measured for a third time), yielding a total set of 95 distinct measurements. BSIMM-V includes 67 firms, 21 of which have been remeasured (with 4 firms measured for a third time), yielding a total set of 161 distinct measurements. As of BSIMM-V, 5 firms (representing 5 distinct measurements) have been dropped because their measurements were more than 48 months old.

BSIMM Community

The 67 firms participating in the BSIMM Project make up the BSIMM Community. A moderated private mailing list with over 200 members allows SSG leaders participating in the BSIMM to discuss solutions with others who face the same issues, discuss strategy with someone who has already addressed an issue, seek out mentors from those are farther along a career path, and band together to solve hard problems.

The BSIMM Community also hosts annual private conferences where up to three representatives from each firm gather together in an off-the-record forum to discuss software security initiatives. In Fall of 2012, 28 of 51 firms participated in the third annual BSIMM Community Conference in Galloway, New Jersey. In Spring of 2013, 10 of 15 firms with a presence in the E.U. participated in the Second Annual BSIMM Europe Community Conference in London, England.

The BSIMM website <http://bsimm.com> includes a credentialed BSIMM Community section where information from the Conferences, working groups, and mailing list-initiated studies are posted.

The Software Security Framework

The table below shows the SSF. There are twelve *practices* organized into four *domains*. The domains are:

1. **Governance**: Practices that help organize, manage, and measure a software security initiative. Staff development is also a central governance practice.
2. **Intelligence**: Practices that result in collections of corporate knowledge used in carrying out software security activities throughout the organization. Collections include both proactive security guidance and organizational threat modeling.
3. **SSDL Touchpoints**: Practices associated with analysis and assurance of particular software development artifacts and processes. All software security methodologies include these practices.
4. **Deployment**: Practices that interface with traditional network security and software maintenance organizations. Software configuration, maintenance, and other environment issues have direct impact on software security.

The Software Security Framework (SSF)			
Governance	**Intelligence**	**SSDL Touchpoints**	**Deployment**
Strategy and Metrics	Attack Models	Architecture Analysis	Penetration Testing
Compliance and Policy	Security Features and Design	Code Review	Software Environment
Training	Standards and Requirements	Security Testing	Configuration Management and Vulnerability Management

There are three practices under each domain. To provide some idea of what a practice entails, we include a short explanation of each. Note that the BSIMM describes objectives and activities for each practice.

In the **governance** domain, the *strategy and metrics* practice encompasses planning, assigning roles and responsibilities, identifying software security goals, determining budgets, and identifying metrics and gates. The *compliance and policy* practice is focused on identifying controls for compliance regimens such as PCI DSS and HIPAA, developing contractual controls such as Service Level Agreements to help control COTS software risk, setting organizational software security policy, and auditing against that policy. *Training* has always played a critical role in software security because software developers and architects often start with very little security knowledge.

The **intelligence** domain is meant to create organization-wide resources. Those resources are divided into three practices. *Attack models* capture information used to think like an attacker: threat modeling, abuse case development and refinement, data classification, and technology-specific attack patterns. The *security features and design* practice is charged with creating usable security patterns for major security controls (meeting the standards defined in the next practice), building middleware frameworks for those controls, and creating and publishing other proactive security guidance. The *standards and requirements* practice involves eliciting explicit security requirements from the organization, determining which COTS to recommend, building standards for major security controls (such as authentication, input validation, and so on), creating security standards for technologies in use, and creating a standards review board.

The **SSDL Touchpoints** domain is probably the most familiar of the four. This domain includes essential software security best practices that are integrated into the SDLC. The two most important software security practices are architecture analysis and code review. *Architecture analysis* encompasses capturing software architecture in concise diagrams, applying lists of risks and threats, adopting a process for review (such as STRIDE or Architectural Risk Analysis), and building an assessment and remediation plan for the organization. The *code review* practice includes use of code review tools, development of tailored rules, customized profiles for tool use by different roles (for example, developers versus auditors), manual analysis, and tracking/measuring results. The *security testing* practice is concerned with pre-release testing including integrating security into standard quality assurance processes. The practice includes use of black box security tools (including fuzz testing) as a smoke test in QA, risk driven white box testing, application of the attack model, and code coverage analysis. Security testing focuses on vulnerabilities in construction.

By contrast, in the **deployment** domain, the *penetration testing* practice involves more standard outside→in testing of the sort carried out by security specialists. Penetration testing focuses on vulnerabilities in final configuration, and provides direct feeds to defect management and mitigation. The *software environment* practice concerns itself with OS and platform patching, web application firewalls, installation and configuration documentation, application monitoring, change management, and ultimately code signing. Finally, the *configuration management and vulnerability management* practice concerns itself with patching and updating applications, version control, defect tracking and remediation, and incident handling.

BSIMM-V

We present the maturity model as a series of activities associated with each of the twelve practices. Our approach is to identify goals for each level of a practice, which can be further split into objectives for the practice/level and are in this way associated with activities. Take a look at the BSIMM skeleton if you need help understanding how the basic activities are organized.

Characterizing and selling an initiative's business goals is part of making a software security initiative a success. From a top-down perspective, a goal-based approach includes identifying initiative-level and domain-level goals as well as goals and objectives for practices, levels, and activities.

To risk repeating ourselves, the top-level goals for the BSIMM writ large are:

- Informed risk management decisions
- Clarity on what is "the right thing to do" for everyone involved in software security
- Cost reduction through standard, repeatable processes
- Improved code quality

Each of the four domains in the SSF has associated goals of its own. By understanding these goals, you can quickly see why adopting some aspects of all four domains makes sense. Certainly, ignoring an entire domain would be folly. Here are the goals associated with each domain in the BSIMM:

Domain	Business Goals
Governance	Transparency, Accountability, Checks and Balances
Intelligence	Auditability, Stewardship, Standardization
SSDL Touchpoints	Quality Control
Deployment	Quality Control, Change Management

In the SSF, there are three practices in each domain. Like domains, each practice has a high-level goal. By understanding these goals, you can see at a glance why adopting some aspects of all twelve practices makes sense. At this level, you can begin to think about which goals are the most important to your organization and its culture. Here are the goals associated with each practice in the BSIMM:

Domain	Practice	Business Goals
Governance	Strategy and Metrics	Transparency of expectations, Accountability for results
	Compliance and Policy	Prescriptive guidance for all stakeholders, Auditability
	Training	Knowledgeable workforce, Error correction
Intelligence	Attack Models	Customized knowledge
	Security Features and Design	Reusable designs, Prescriptive guidance for all stakeholders
	Standards and Requirements	Prescriptive guidance for all stakeholders
SSDL Touchpoints	Architecture Analysis	Quality control
	Code Review	Quality control
	Security Testing	Quality control
Deployment	Penetration Testing	Quality control
	Software Environment	Change management
	Configuration Management and Vulnerability Management	Change management

GOVERNANCE: Strategy and Metrics (SM)

The overall goals for the Strategy and Metrics practice are transparency of expectations and accountability for results. Executive management must clarify organizational expectations for the SSDL so that everyone understands the importance of the initiative. In addition, executive management must set specific objectives for all SSDL stakeholders and ensure that specific individuals are made accountable for meeting those objectives.

one

SM Level 1: Attain a common understanding of direction and strategy. Managers must ensure that everyone associated with creating, deploying, operating, and maintaining software understands the written organizational software security objectives. Leaders must also ensure that the organization as a whole understands the strategy for achieving these objectives. A common strategic understanding is essential for effective and efficient program execution.

[SM1.1] Publish process (roles, responsibilities, plan), evolve as necessary. The process for addressing software security is broadcast to all participants so that everyone knows the plan. Goals, roles, responsibilities, and activities are explicitly defined. Most organizations pick and choose from a published methodology such as the Microsoft SDL or the Cigital Touchpoints and then tailor the methodology to their needs. An SSDL process evolves as the organization matures and as the security landscape changes. In many cases, the methodology is published only internally and is controlled by the SSG. The SSDL does not need to be publically promoted outside of the firm to count.

[SM1.2] Create evangelism role and perform internal marketing. In order to build support for software security throughout the organization, someone in the SSG plays an evangelism role. This internal marketing function helps keep the organization current on the magnitude of the software security problem and the elements of its solution. Evangelists might give talks for internal groups, extend invitations to outside speakers, author white papers for internal consumption, or create a collection of papers, books, and other resources on an internal web site and promote its use. Ad hoc conversations between the SSG and executives or an SSG where "everyone is an evangelist" does not achieve the desired results. A canonical example of such an evangelist is Michael Howard's role at Microsoft.

[SM1.3] Educate executives. Executives learn about the consequences of inadequate software security and the negative business impact that poor security can have. They also learn what other organizations are doing to attain software security. By understanding both the problem and its proper resolution, executives come to support the software security initiative as a risk management necessity. In its most dangerous form, such education arrives courtesy of malicious hackers or public data exposure incidents. Preferably, the SSG demonstrates a worst-case scenario in a controlled environment with the permission of all involved (e.g., actually showing working exploits and their business impact). In some cases, presentation to the Board can help garner resources for an ongoing software security initiative. Bringing in an outside guru is often helpful when seeking to bolster executive attention.

[SM1.4] Identify gate locations, gather necessary artifacts. The software security process will involve release gates/checkpoints/milestones at one or more points in the SDLC or, more likely, the SDLCs. The first two steps toward establishing release gates are: 1) to identify gate locations that are compatible with existing development practices and 2) to begin gathering the input necessary for making a go/no go decision. Importantly at this stage, the gates are not enforced. For example, the SSG can collect security testing results for each project prior to release, but stop short of passing judgment on what constitutes sufficient testing or acceptable test results. The idea of identifying gates first and only enforcing them later is extremely helpful in moving development toward software security without major pain. Socialize the gates, and only turn them on once most projects already know how to succeed. This gradual approach serves to motivate good behavior without requiring it.

[SM1.6] Require security sign-off. The organization has an initiative-wide process for accepting security risk and documenting accountability. A risk acceptor signs off on the state of all software prior to release. For example, the sign-off policy might require the head of the business unit to sign off on critical vulnerabilities that have not been mitigated or SSDL steps that have been skipped. Informal risk acceptance alone does not count as security sign off, as the act of accepting risk is more effective when it is formalized (e.g., with a signature, form submission, or similar) and captured for future reference.

two

SM Level 2: Align behavior with strategy and verify adherence. Managers must explicitly identify individuals responsible for software security risk management accountability. These individuals, are in turn responsible for ensuring successful performance of SSDL activities. SSDL managers must ensure quick identification and modification of any SSDL behavior resulting in unacceptable risk. To reduce unacceptable risk, managers must identify and encourage the growth of a software security satellite (see the *Roles* section above).

[SM2.1] Publish data about software security internally. The SSG publishes data internally on the state of software security within the organization. The information might come as a dashboard with metrics for executives and software development management. Sometimes, publication is not shared with everyone in a firm, but rather with the relevant executives only. Publishing information up to executives who then do something about it and drive change in the organization suffices. In other cases, open book management and publishing data to all stakeholders helps everyone know what's going on, with the philosophy that sunlight is the best disinfectant. If the organization's culture promotes internal competition between groups, this information adds a security dimension to the game.

[SM2.2] Enforce gates with measurements and track exceptions. SDLC security gates are now enforced: in order to pass a gate, a project must either meet an established measure or obtain a waiver. Even recalcitrant project teams must now play along. The SSG tracks exceptions. A gate could require a project to undergo code review and remediate any critical findings before release. In some cases, gates are directly associated with controls required by regulations, contractual agreements, and other business obligations and exceptions are tracked as required by statutory or regulatory drivers. In other cases, gate measures yield key performance indicators that are used to govern the process.

[SM2.3] Create or grow a satellite. The satellite begins as a collection of people scattered across the organization who show an above-average level of security interest or skill. Identifying this group is a step towards creating a social network that speeds the adoption of security into software development. One way to begin is to track the people who stand out during introductory training courses. (See *[T2.7 Identify satellite through training]*.) Another way is to ask for volunteers. In a more top down approach, initial satellite membership is assigned to ensure complete coverage of all development/product groups. Ongoing membership should be based on actual performance. A strong satellite is a good sign of a mature software security initiative.

[SM2.5] Identify metrics and use them to drive budgets. The SSG and its management choose the metrics that define and measure software security initiative progress. These metrics will drive the initiative's budget and allocation of resources so simple counts and statistics won't suffice. Metrics also allow the SSG to explain its goals and its progress in quantitative terms. One such metric could be security defect density. A reduction in security defect density could be used to show a decreasing cost of remediation over time. The key here is to tie technical results to business objectives in a clear and obvious fashion in order to justify funding. Since the concept of security is already tenuous to business people, making this explicit tie can be very helpful.

three

SM Level 3: Practice risk-based portfolio management. Application owners and the SSG must inform management of the risk associated with each application in the portfolio. The SSG must advertise its activities externally to create support for its approach and enable ecosystem security.

[SM3.1] Use an internal tracking application with portfolio view. The SSG uses a centralized tracking application to chart the progress of every piece of software in its purview. The application records the security activities scheduled, in progress, and completed. It incorporates results from activities such as architecture analysis, code review, and security testing. The SSG uses the tracking application to generate portfolio reports for many of the metrics it uses. A combined inventory and risk posture view is fundamental. In many cases, these data are published at least among executives. Depending on the culture, this can cause interesting effects through internal competition. As an initiative matures, and activities become more distributed, the SSG uses the centralized reporting system to keep track of all of the moving parts.

[SM3.2] Run an external marketing program. The SSG markets the software security initiative outside the firm to build external support. Software security grows beyond being a risk reduction exercise and becomes a competitive advantage or market differentiator. The SSG might write papers or books. It might have a public blog. It might participate in external conferences or trade shows. In some cases, a complete SSDL methodology can be published and promoted externally. Sharing details externally and inviting critique can bring new perspectives into the firm.

GOVERNANCE: Compliance and Policy (CP)

The overall goals for the Compliance and Policy practice are prescriptive guidance for all stakeholders and auditability of SSDL activities. Management-approved prescriptive guidance must be available to all SSDL stakeholders, including vendors, for use in meeting security and compliance objectives. All SSDL activities must produce artifacts sufficient to allow auditing for adherence to prescriptive guidance.

one

CP Level 1: Document and unify statutory, regulatory, and contractual compliance drivers. The SSG must work with appropriate groups to capture unified compliance requirements in prescriptive guidance and make that knowledge available to SSDL stakeholders.

[CP1.1] Unify regulatory pressures. If the business or its customers are subject to regulatory or compliance drivers such as FFIEC, GLBA, OCC, PCI DSS, SOX, HIPAA or others, the SSG acts as a focal point for understanding the constraints such drivers impose on software. In some cases, the SSG creates a unified approach that removes redundancy from overlapping compliance requirements. A formal approach will map applicable portions of regulations to control statements explaining how the organization complies. As an alternative, existing business processes run by legal or other risk and compliance groups outside the SSG may also serve as the regulatory focal point. The goal of this activity is to create one set of software security guidance so that compliance work is completed as efficiently as possible (mostly by removing duplicates). Some firms move on to guide exposure by becoming directly involved in standards groups in order to influence the regulatory environment.

[CP1.2] Identify PII obligations. The way software handles personally identifiable information (PII) could well be explicitly regulated, but even if it is not, privacy is a hot topic. The SSG takes a lead role in identifying PII obligations stemming from regulation and customer expectations. It uses this information to promote best practices related to privacy. For example, if the organization processes credit card transactions, the SSG will identify the constraints that the PCI DSS places on the handling of cardholder data. Note that outsourcing to hosted environments (e.g., the cloud) does not relax a majority of PII obligations. Also note, firms that create software products that process PII (but don't necessarily handle PII directly) may provide privacy controls and guidance for their customers.

[CP1.3] Create policy. The SSG guides the rest of the organization by creating or contributing to software security policy that satisfies regulatory and customer-driven security requirements. The policy provides a unified approach for satisfying the (potentially lengthy) list of security drivers at the governance level. As a result, project teams can avoid learning the details involved in complying with all applicable regulations. Likewise, project teams don't need to re-learn customer security requirements on their own. The SSG policy documents are sometimes focused around major compliance topics such as the handling of PII or the use of cryptography. In some cases, policy documents relate directly to the SSDL and its use in the firm. Architecture standards and coding guidelines are not examples of software security policy. On the other hand, policy that prescribes and mandates the use of coding guidelines and architecture standards for certain categories of applications does count. Policy is what is permitted and denied at the initiative level.

two

CP Level 2: Align internal practices with compliance drivers and policy, backed by executives. Executives must overtly promote the SSG and associated software security initiative, including the need for compliance. Risk managers must explicitly take responsibility for software risk. The SSG and application owners must ensure service level agreements address security properties of vendor software deliverables.

[CP2.1] Identify PII data inventory. The organization identifies the kinds of PII stored by each of its systems and their data repositories. A PII inventory can be approached in two ways: through each individual application by noting its PII use, or through particular types of PII and the applications that touch them. When combined with the organization's PII obligations, this inventory guides privacy planning. For example, the SSG can now create a list of databases that would require customer notification if breached.

[CP2.2] Require security sign-off for compliance-related risk. The organization has a formal compliance risk acceptance and accountability process addressing all software development projects. The risk acceptor signs off on the state of the software prior to release. For example, the sign-off policy might require the head of the business unit to sign off on compliance issues that have not been mitigated or SSDL steps related to compliance that have been skipped. Sign off should be explicit and captured for future reference, and exceptions should be tracked.

[CP2.3] Implement and track controls for compliance. The organization can demonstrate compliance with applicable regulations because its practices are aligned with the control statements developed by the SSG. (See *[CP1.1 Unify regulatory pressures]*.) The SSG tracks the controls, shepherds problem areas, and makes sure auditors and regulators are satisfied. If the organization's SDLC is predictable and reliable, the SSG might be able to largely sit back and keep score. If the SDLC is uneven or less reliable, the SSG could be forced to take a more active role as referee. A firm doing this properly can explicitly tie satisfying its compliance concerns to its SSDL.

[CP2.4] Paper all vendor contracts with software security SLAs. Vendor contracts include a service-level agreement (SLA) ensuring that the vendor will not jeopardize the organization's compliance story and software security initiative. Each new or renewed contract contains a set of provisions requiring the vendor to deliver a product or service compatible with the organization's security policy. (See *[SR2.5 Create SLA boilerplate]*.) In some cases, open source licensing concerns initiate the vendor control process. That can open the door for further software security language in the SLA.

[CP2.5] Promote executive awareness of compliance and privacy obligations. The SSG gains executive buy-in around compliance and privacy activities. Executives are provided plain-language explanations of the organization's compliance and privacy obligations and the potential consequences for failing to meet those obligations. For some organizations, explaining the direct cost and likely fallout from a data breach could be an effective way to broach the subject. For other organizations, having an outside expert address the Board works because some executives value outside perspective over internal perspective. One sure sign of proper executive awareness is adequate allocation of resources to get the job done.

three

CP Level 3: Organizational threat, attack, defect, and operational issue data drive policy evolution and demands on vendors. Executives must ensure that software security policy is periodically updated based on actual data and must demonstrate the organization's ongoing compliance. The SSG, application owners, and legal groups must ensure vendors deliver software that complies with relevant organizational policy.

[CP3.1] Create regulator eye-candy. The SSG has the information regulators want. A combination of written policy, controls documentation, and artifacts gathered through the SSDL gives the SSG the ability to demonstrate the organization's compliance story without a fire drill for every audit. In some cases, regulators, auditors, and senior management are satisfied with the same kinds of reports, which may be generated directly from various tools.

[CP3.2] Impose policy on vendors. Vendors are required to adhere to the same policies used internally. Vendors must submit evidence that their software security practices pass muster. Evidence could include code review results or penetration test results. Vendors may also attest to the fact that they are carrying out certain SSDL processes. In some cases, a BSIMM score or a vBSIMM score has been used to help ensure that vendors are complying with the firm's policies.

[CP3.3] Drive feedback from SSDL data back to policy. Information from the SSDL is routinely fed back into the policy creation process. Policies are improved to find defects earlier or prevent them from occurring in the first place. Blind spots are eliminated based on trends in SSDL failures. For example, inadequate architecture analysis, recurring vulnerabilities, ignored security gates, or choosing the wrong firm to carry out a penetration test may expose policy weakness. Over time, policies should become more practical and easier to carry out. (See *[SM1.1 Publish process (roles, responsibilities, plan), evolve as necessary]*.) Ultimately policies align themselves with the SSDL data and enhance and improve a firm's effectiveness.

GOVERNANCE: Training (T)

The overall goals for the Training practice are the creation of a knowledgeable workforce and correcting errors in processes. The workforce must have role-based knowledge that specifically includes the skills required to adequately perform their SSDL activities. Training must include specific information on root causes of errors discovered in process activities and outputs.

one

T Level 1: Make customized, role-based training available on demand. The SSG must build interest in software security throughout the organization and provide role-specific training material, including computer-based training, that incorporates lessons from actual internal events.

[T1.1] Provide awareness training. The SSG provides awareness training in order to promote a culture of software security throughout the organization. Training might be delivered by members of the SSG, by an outside firm, by the internal training organization, or through a computer-based training system. Course content is not necessarily tailored for a specific audience. For example, all programmers, quality assurance engineers, and project managers could attend the same *Introduction to Software Security* course. This common activity can be enhanced with a tailored approach to an introductory course that addresses a firm's culture explicitly. Generic introductory courses covering basic IT security and high level software security concepts do not generate satisfactory results. Likewise, providing awareness training only to developers and not to other roles is also insufficient.

[T1.5] Deliver role-specific advanced curriculum (tools, technology stacks, bug parade). Software security training goes beyond building awareness and enables trainees to incorporate security practices into their work. The training is tailored to the role of trainees; trainees get information on the tools, technology stacks, or kinds of bugs that are most relevant to them. An organization might offer four tracks for engineers: one for architects, one for Java developers, one for .NET. developers, and a fourth for testers. Tool-specific training is also commonly observed in a curriculum. Don't forget that training will be useful for many different roles in an organization, including QA, product management, executives, and others.

[T1.6] Create and use material specific to company history. In order to make a strong and lasting change in behavior, training includes material specific to the company's history. When participants can see themselves in the problem, they are more likely to understand how the material is relevant to their work and to know when and how to apply what they have learned. One way to do this is to use noteworthy attacks on the company as examples in the training curriculum. Be wary of training that covers platforms not used by developers (Windows developers don't care about old Unix problems) or examples of problems only relevant to languages no longer in common use (Java developers don't need to understand buffer overflows in C). Stories from company history can help steer training in the right direction only if the stories are still relevant.

[T1.7] Deliver on-demand individual training. The organization lowers the burden on trainees and reduces the cost of delivering training by offering on-demand training for individuals. Computer-based training (CBT) is the most obvious choice and can be kept up to date through a subscription model. CBT courses must be engaging and relevant to achieve their intended purpose. For developers, it is also possible to provide training directly through IDEs right at the time it's needed. Remember that in some cases, building a new skill (such as code review) may be better suited for instructor-led training.

two

T Level 2: Create the software security satellite. The SSG must build and enhance a satellite through social activities, including training and related events. The SSG and managers must ensure that new hires are exposed to the corporate security culture during onboard activities.

[T2.5] Enhance satellite through training and events. The SSG strengthens its social network by holding special events for the satellite. The satellite learns about advanced topics or hears from guest speakers. Offering pizza and beer doesn't hurt. A standing conference call meeting does not address this activity, which is as much about building camaraderie as it is about sharing knowledge or organizational efficiency. There is no substitute for face-to-face meetings, even if they happen only once or twice a year.

[T2.6] Include security resources in onboarding. The process for bringing new hires into the engineering organization requires a module on software security. The generic new hire process covers things like picking a good password and making sure people don't tail you into the building, but this is enhanced further to cover topics such as secure coding, the SSDL, and internal security resources. The objective is to ensure that new hires enhance the security culture. Turnover in engineering organizations is generally high. Though a generic onboarding module is useful, it does not take the place of a timely and more complete introductory software security course.

[T2.7] Identify satellite through training. The satellite begins as a collection of people scattered across the organization who show an above-average level of security interest or skill. Identifying this group is a step towards creating a social network that speeds the adoption of security into software development. One way to begin is to track the people who stand out during training courses. (See *[SM2.3 Create or grow a satellite]*.) In general, a volunteer army may be easier to lead than one that is drafted.

three

T Level 3: Provide recognition for skills and career path progression. Also build morale. Management and the SSG must ensure that all staff members receive appropriate recognition for advancement through the training curriculum. Managers, application owners, and the SSG must provide training to vendors and outsource workers as a method of spreading the security culture. Managers and the SSG must continue to bolster satellite momentum by marketing the security culture externally. The SSG must be available, at least periodically, for those seeking software security guidance. Managers must ensure that all staff members receive this training at least annually.

[T3.1] Reward progression through curriculum (certification or HR). Knowledge is its own reward, but progression through the security curriculum brings other benefits too. Developers and testers see a career advantage in learning about security. The reward system can be formal and lead to a certification or official mark in the HR system, or it can be less formal and make use of motivators such as praise letters for the satellite written just before annual review time. Involving a corporate training department and/or HR can make security's impact on career progression more obvious, but the SSG should continue to monitor security knowledge in the firm and not cede complete control or oversight.

[T3.2] Provide training for vendors or outsourced workers. The organization delivers security training for vendors and outsource providers. Spending time and effort helping suppliers get security right is easier than trying to figure out what they screwed up later on. In the best case, outsourced workers receive the same training given to employees. Training individual contractors is much more natural than training entire outsource firms and is a reasonable way to start. Of course, it is important to train everyone who works on your software regardless of their employment status.

[T3.3] Host external software security events. The organization highlights its security culture as a differentiator by hosting external security events. Microsoft's BlueHat is such an event, as is Intel's Security Conference. Employees benefit from hearing outside perspectives. The organization as a whole benefits from putting its security cred on display. (See *[SM3.2 Run an external marketing program]*.)

[T3.4] Require an annual refresher. Everyone involved in making software is required to take an annual software security refresher course. The refresher keeps the staff up-to-date on security and ensures the organization doesn't lose focus due to turnover. The SSG might use half a day to give an update on the security landscape and explain changes to policies and standards. A refresher can be rolled out as part of a firm-wide security day or in concert with an internal security conference.

[T3.5] Establish SSG office hours. The SSG offers help to any and all comers during an advertised lab period or regularly scheduled office hours. By acting as an informal resource for people who want to solve security problems, the SSG leverages teachable moments and emphasizes the carrot over the stick. Office hours might be held one afternoon per week in the office of a senior SSG member. Mobile office hours are also a possibility, with visits to particular product or application groups slated by request.

INTELLIGENCE: Attack Models (AM)

The overall goal for the Attack Models practice is the creation of customized knowledge on attacks relevant to the organization. Customized knowledge must guide decisions about both code and controls.

one

AM Level 1: Create attack and data asset knowledge base. The SSG must identify potential attackers and document both the attacks that cause the greatest organizational concern and any important attacks that have already occurred. The SSG must communicate attacker information to all interested parties. The business must create a data classification scheme that the SSG uses to inventory and prioritize applications.

[AM1.1] Build and maintain a top N possible attacks list. The SSG helps the organization understand attack basics by maintaining a list of attacks most important to the firm and using it to drive change. This list combines input from multiple sources: observed attacks, hacker forums, industry trends, etc. The list does not need to be updated with great frequency and the attacks can be sorted in a coarse fashion. For example, the SSG might brainstorm twice per year to create lists of attacks the organization should be prepared to counter "now," "soon," and "someday." In some cases, attack model information is used in a list-based approach to architecture analysis, helping to focus the analysis as in the case of STRIDE.

[AM1.2] Create a data classification scheme and inventory. The organization agrees upon a data classification scheme and uses the scheme to inventory its software according to the kinds of data the software handles. This allows applications to be prioritized by their data classification. Many classification schemes are possible—one approach is to focus on PII. Depending upon the scheme and the software involved, it could be easiest to first classify data repositories, then derive classifications for applications according to the repositories they use. Other approaches to the problem are possible. For example, data may be classified according to protection of intellectual property, impact of disclosure, exposure to attack, relevance to SOX, or geographic boundaries.

[AM1.3] Identify potential attackers. The SSG identifies potential attackers in order to understand their motivations and capabilities. The outcome of this exercise could be a set of attacker profiles including generic sketches for broad categories of attackers and more detailed descriptions for noteworthy individuals. In some cases, a third-party vendor may be contracted to provide this information. Specific and contextual attacker information is almost always more useful than generic information copied from someone else's list.

[AM1.4] Collect and publish attack stories. In order to maximize the benefit from lessons that do not always come cheap, the SSG collects and publishes stories about attacks against the organization. Over time, this collection helps the organization understand its history. Both successful and unsuccessful attacks can be noteworthy. Discussing historical information about software attacks has the effect of grounding software security in the reality of a firm. This is particularly useful in training classes in order to counter a generic approach over-focused on top ten lists or irrelevant and outdated platform attacks. Hiding information about attacks from people building new systems does nothing to garner positive benefit from a negative happenstance.

[AM1.5] Gather attack intelligence. The SSG stays ahead of the curve by learning about new types of attacks and vulnerabilities. The information comes from attending conferences and workshops, monitoring attacker forums, and reading relevant publications, mailing lists, and blogs. Make Sun Tzu proud by knowing your enemy; engage with the security researchers who are likely to cause you trouble. In many cases, a subscription to a commercial service provides a reasonable way of gathering basic attack intelligence. Regardless of its origin, attack information must be made actionable and useful for software builders and testers.

[AM1.6] Build an internal forum to discuss attacks. The organization has an internal forum where the SSG, the satellite, and others discuss attacks. The forum serves to communicate the attacker perspective. The SSG could maintain an internal mailing list where subscribers share the latest information on publicly known incidents. Dissection of attacks and exploits that are relevant to a firm can be particularly helpful, especially if they spur discussion of development mitigations. Simply republishing items from public mailing lists does not achieve the same benefits as active discussion. Vigilance means never getting too comfortable. (See *[SR1.2 Create a security portal]*.)

two

AM Level 2: Provide outreach on attackers and relevant attacks. The SSG must gather attack intelligence and expand its attack knowledge to include both higher-level attack patterns and lower-level abuse cases. Attack patterns must include technology-specific information relevant to the organization.

[AM2.1] Build attack patterns and abuse cases tied to potential attackers. The SSG prepares for security testing and architecture analysis by building attack patterns and abuse cases tied to potential attackers. These resources do not have to be built from scratch for every application in order to be useful. Instead, there could be standard sets for applications with similar profiles. The SSG will add to the pile based on attack stories. For example, a story about an attack against poorly managed entitlements could lead to an entitlements attack pattern that drives a new type of testing. If a firm tracks fraud and monetary costs associated with particular attacks, this information can be used to guide the process of building attack patterns and abuse cases.

[AM2.2] Create technology-specific attack patterns. The SSG creates technology-specific attack patterns to capture knowledge about attacks that target particular technologies. For example, if the organization's web software relies on cutting-edge browser capabilities, the SSG could catalogue the quirks of all the popular browsers and how they might be exploited. Attack patterns directly related to the security frontier (currently mobile security and cloud security) may be useful. Simply republishing general guidelines (e.g., "Ensure data are protected in transit") and adding "for mobile applications" on the end does not constitute technology-specific attack patterns.

three

AM Level 3: Research and mitigate new attack patterns. The SSG must conduct attack research on corporate software to get ahead of attacker activity. The SSG must provide knowledge and automation to auditors and testers to ensure their activities reflect actual attacks perpetrated against the organization's software as well as potential attacks.

[AM3.1] Have a science team that develops new attack methods. The SSG has a science team that works to identify and defang new classes of attacks before real attackers even know they exist. This is not a penetration testing team finding new instances of known types of weaknesses—it is a research group innovating new types of attacks. A science team may include well-known security researchers who publish their findings at conferences like Def Con.

[AM3.2] Create and use automation to do what attackers will do. The SSG arms testers and auditors with automation to do what attackers are going to do. For example, a new attack method identified by the science team could require a new tool. The SSG packages the new tool and distributes it to testers. The idea here is to push attack capability past what typical commercial tools and offerings encompass and then package that information for others to use. Tailoring tools to a firm's particular technology stacks and potential attackers is a really good idea.

INTELLIGENCE: Security Features and Design (SFD)

The overall goal for the Security Features and Design practice is the creation of customized knowledge on security features, frameworks, and patterns. The customized knowledge must drive architecture and component decisions.

one

SFD Level 1: Publish security features and architecture. The SSG must provide architects and developers with guidance on security features and participate directly with architecture groups.

[SFD1.1] Build and publish security features. Some problems are best solved only once. Rather than have each project team implement all of their own security features (authentication, role management, key management, audit/log, cryptography, protocols), the SSG provides proactive guidance by building and publishing security features for other groups to use. Project teams benefit from implementations that come pre-approved by the SSG, and the SSG benefits by not having to repeatedly track down the kinds of subtle errors that often creep into security features. The SSG can identify an implementation they like and promote it as the accepted solution.

[SFD1.2] Engage SSG with architecture. Security is a regular part of the organization's software architecture discussion. The architecture group takes responsibility for security the same way they take responsibility for performance, availability, or scalability. One way to keep security from falling out of the discussion is to have an SSG member attend regular architecture meetings. In other cases, enterprise architecture can help the SSG create secure designs that integrate properly into corporate design standards.

two

SFD Level 2: Build and identify security solutions. The SSG must provide secure-by-design frameworks and must be available for and capable of solving design problems for others.

[SFD2.1] Build secure-by-design middleware frameworks and common libraries. The SSG takes a proactive role in software design by building or providing pointers to secure-by-design middleware frameworks or common libraries. In addition to teaching by example, this middleware aids architecture analysis and code review because the building blocks make it easier to spot errors. For example, the SSG could modify a popular web framework such as Spring to make it easy to meet input validation requirements. Eventually the SSG can tailor code review rules specifically for the components it offers. (See *[CR3.1 Use automated tools with tailored rules]*.) When adopting a middleware framework (or any other widely used software), careful vetting for security before publication is important. Encouraging adoption and use of insecure middleware does not help the software security situation. Generic open source software security architectures, including OWASP ESAPI, should not be considered secure out of the box.

[SFD2.2] Create SSG capability to solve difficult design problems. When the SSG is involved early in the new project process, it contributes to new architecture and solves difficult design problems. The negative impact security has on other constraints (time to market, price, etc.) is minimized. If an architect from the SSG is involved in the design of a new protocol, he or she could analyze the security implications of existing protocols and identify elements that should be duplicated or avoided. Designing for security up front is more efficient than analyzing an existing design for security and then refactoring when flaws are uncovered. Some design problems will require specific expertise outside of the SSG.

three

SFD Level 3: Actively reuse approved security features and secure-by-design frameworks. The SSG must provide additional mature design patterns taken from existing software and technology stacks. Managers must ensure there is formal consensus across the organization on secure design choices. Managers must also require that defined security features and frameworks be used whenever possible.

[SFD3.1] Form a review board or central committee to approve and maintain secure design patterns. A review board or central committee formalizes the process for reaching consensus on design needs and security tradeoffs. Unlike the architecture committee, this group is specifically focused on providing security guidance. The group can also periodically review already-published design standards (especially around cryptography) to ensure that design decisions do not become stale or out of date.

[SFD3.2] Require use of approved security features and frameworks. Implementers must take their security features and frameworks from an approved list. There are two benefits: developers do not spend time re-inventing existing capabilities, and review teams do not have to contend with finding the same old defects in brand new projects. In particular, the more a project uses proven components, the easier architecture analysis and code review become. (See *[AA1.1Perform security feature review]*.) Re-use is a major advantage of consistent software architecture.

[SFD3.3] Find and publish mature design patterns from the organization. The SSG fosters centralized design reuse by finding and publishing mature design patterns from and throughout the organization. A section of the SSG web site could promote positive elements identified during architecture analysis so that good ideas are spread. This process should be formalized. An ad hoc, accidental noticing is not sufficient. In some cases, a central architecture or technology team facilitates and enhances this activity.

INTELLIGENCE: Standards and Requirements (SR)

The overall goal for the Standards and Requirements practice is to create prescriptive guidance for all stakeholders. Managers and the SSG must document software security choices and convey this material to everyone involved in the SSDL, including external parties.

one

SR Level 1: Provide easily accessible security standards and requirements. The SSG must provide foundational knowledge, including at the very least: security standards, secure coding standards, and compliance requirements. Managers must ensure that software security information is kept up-to-date and made available to everyone.

[SR1.1] Create security standards. Software security requires much more than security features, but security features are part of the job as well. The SSG meets the organization's demand for security guidance by creating standards that explain the accepted way to adhere to policy and carry out specific security-centric operations. A standard might describe how to perform authentication using J2EE or how to determine the authenticity of a software update. (See *[SFD1.1 Build and publish security features]* for one case where the SSG provides a reference implementation of a security standard.) Standards can be deployed in a variety of ways. In some cases, standards and guidelines can be automated in development environments (e.g., worked into an IDE). In other cases, guidelines can be explicitly linked to code examples to make them more actionable and relevant.

[SR1.2] Create a security portal. The organization has a central location for information about software security. Typically this is an internal web site maintained by the SSG. People refer to the site for the latest and greatest on security standards and requirements as well as other resources provided by the SSG. An interactive wiki is better than a static portal with guideline documents that only rarely change. Organizations can enhance these materials with mailing lists and face-to-face meetings.

[SR1.3] Translate compliance constraints to requirements. Compliance constraints are translated into software requirements for individual projects. This is a linchpin in the organization's compliance strategy—by representing compliance constraints explicitly with requirements, demonstrating compliance becomes a manageable task. For example, if the organization routinely builds software that processes credit card transactions, PCI DSS compliance could play a role in the SSDL during the requirements phase. In other cases, technology standards built for international interoperability reasons can include security guidance. Representing these standards as requirements helps with traceability and visibility in the case of audit.

[SR1.4] Use secure coding standards. Secure coding standards help developers avoid the most obvious bugs and provide ground rules for code review. Secure coding standards are necessarily specific to a programming language and can address the use of popular frameworks and libraries. If the organization already has coding standards for other purposes, the secure coding standards should build upon them. A clear set of secure coding standards is a good way to guide both manual and automated code review, as well as beefing up security training with relevant examples.

two

SR Level 2: Communicate formally-approved standards internally and to vendors. Managers must ensure that a formal process is used to create standards specific to technology stacks. Managers, the SSG, and product owners must ensure that SLAs are reinforced by contractual language approved by legal staff. The SSG must ensure that all open source software is identified in the organization's code.

[SR2.2] Create a standards review board. The organization creates a standards review board to formalize the process used to develop standards and ensure that all stakeholders have a chance to weigh in. The review board could operate by appointing a champion for any proposed standard. The onus is on the champion to demonstrate that the standard meets its goals and to get approval and buy-in from the review board. Enterprise architecture or enterprise risk groups sometimes take on the responsibility of creating and managing standards review boards.

[SR2.3] Create standards for technology stacks. The organization standardizes on specific technology stacks. For the SSG, this means a reduced workload because the group does not have to explore new technology risks for every new project. Ideally, the organization will create a secure base configuration for each technology stack, further reducing the amount of work required to use the stack safely. A stack might include an operating system, a database, an application server, and a runtime environment for a managed language. The security frontier is a good place to find traction. Currently, mobile technology stacks and platforms as well as cloud-based technology stacks are two areas where specific attention to security pays off.

[SR2.4] Identify open source. The first step toward managing risk introduced by open source is to identify the open source components in use. It is not uncommon to discover old versions of components with known vulnerabilities or multiple versions of the same component. Automated tools for finding open source are one way to approach this activity. A process that relies solely on developers asking for permission does not generate satisfactory results. At the next level of maturity, this activity is subsumed by a policy constraining the use of open source.

[SR2.5] Create SLA boilerplate. The SSG works with the legal department to create standard SLA boilerplate for use in contracts with vendors and outsource providers. The legal department understands that the boilerplate helps prevent compliance and privacy problems. Under the agreement, vendors and outsource providers must meet company software security standards. (See *[CP2.4 Paper all vendor contracts with software security SLAs]*.) Boilerplate language may call out software security vendor control solutions such as vBSIMM measurements or BSIMM scores.

three

SR Level 3: Require risk management decisions for open source use. Managers and the SSG must show that any open source code used in the organization is subject to the same risk management processes as code created internally and ensure that all applicable standards are communicated to third-party vendors.

[SR3.1] Control open source risk. The organization has control over its exposure to the vulnerabilities that come along with using open source components. Use of open source could be restricted to pre-defined projects. It could also be restricted to open source versions that have been through an SSG security screening process, had unacceptable vulnerabilities remediated, and made available only through internal repositories. Legal often spearheads additional open source controls due to the "viral" license problem associated with GPL code. Getting legal to understand security risks can help move an organization to practice decent open source hygiene.

[SR3.2] Communicate standards to vendors. The SSG works with vendors to educate them and promote the organization's security standards. A healthy relationship with a vendor cannot be guaranteed through contract language alone. The SSG engages with vendors, discusses the vendor's security practices, and explains in concrete terms (rather than legalese) what the organization expects of the vendor. Any time a vendor adopts the organization's security standards, it's a clear win. When a firm's SSDL is available publically, communication regarding software security expectations is easier. Likewise, sharing internal practices and measures (including a BSIMM measurement) can make expectations very clear.

SSDL TOUCHPOINTS: Architecture Analysis (AA)

The overall goal of the Architecture Analysis practice is quality control. Those performing architecture analysis must ensure the detection and correction of security flaws. Software architects must enforce adherence to standards and the reuse of approved security features.

one

AA Level 1: Perform risk-driven AA reviews, led by the SSG. The organization must provide a lightweight software risk classification. The SSG must begin leading architecture analysis efforts, particularly on high-risk applications, as a way to build internal capability and demonstrate value at the design level.

[AA1.1] Perform security feature review. To get started with architecture analysis, center the process on a review of security features. Security-aware reviewers first identify the security features in an application (authentication, access control, use of cryptography, etc.) then study the design looking for problems that would cause these features to fail at their purpose or otherwise prove insufficient. For example, a system that was subject to escalation of privilege attacks because of broken access control or a system that stored unsalted password hashes would both be identified in this kind of review. At higher levels of maturity, this activity is eclipsed by a more thorough approach to architecture analysis not centered on features. In some cases, use of the firm's secure-by-design components can streamline this process.

[AA1.2] Perform design review for high-risk applications. The organization learns about the benefits of architecture analysis by seeing real results for a few high-risk, high-profile applications. The reviewers must have some experience performing architecture analysis and breaking the architecture being considered. If the SSG is not yet equipped to perform an in-depth architecture analysis, it uses consultants to do this work. Ad hoc review paradigms that rely heavily on expertise may be used here, though in the long run they do not scale.

[AA1.3] Have SSG lead design review efforts. The SSG takes a lead role in performing architecture analysis in order to begin building the organization's ability to uncover design flaws. Breaking an architecture is enough of an art that the SSG needs to be proficient at it before they can turn the job over to the architects and proficiency requires practice. The SSG cannot be successful on its own either—they will likely need help from the architects or implementers in order to understand the design. With a clear design in hand, the SSG might carry out the analysis with a minimum of interaction with the project team. At higher levels of maturity, the responsibility for leading review efforts shifts towards software architects. Approaches to architecture analysis (and threat modeling) evolve over time. Do not expect to set a process and use it forever.

[AA1.4] Use a risk questionnaire to rank applications. To facilitate the architecture analysis and other processes, the SSG uses a risk questionnaire to collect basic information about each application so that it can determine a risk classification and prioritization scheme. Questions might include, "Which programming languages is the application written in?," "Who uses the application?," and "Does the application handle PII?" A qualified member of the application team completes the questionnaire. The questionnaire is short enough to be completed in a matter of hours. The SSG might use the answers to bucket the application as high, medium, or low risk. Because a risk questionnaire can be easy to game, it is important that some spot checking for validity and accuracy be put in place. An over-reliance on self-reporting or automation can render this activity impotent.

two

AA Level 2: Provide outreach on use of documented AA process. The SSG must facilitate organization-wide use of architecture analysis by making itself available as a resource and mentor. The SSG must define an architecture analysis process based on a common architecture description language and standard attack models.

[AA2.1] Define and use AA process. The SSG defines and documents a process for performing architecture analysis and applies it in the design reviews it conducts. The process includes a standardized approach for thinking about attacks and security properties. The process is defined rigorously enough that people outside the SSG can be taught to carry it out. Particular attention should be paid to documentation of both the architecture under review and any security flaws uncovered. Tribal knowledge does not count as a defined process. Microsoft's

STRIDE and Cigital's ARA are examples of such a process. Note that even these two methodologies for architecture analysis have evolved greatly over time. Make sure to access up-to-date sources for architecture analysis information as many early publications are outdated and no longer apply.

[AA2.2] Standardize architectural descriptions (including data flow). The organization uses an agreed-upon format for describing architecture, including a means for representing data flow. This format, combined with an architecture analysis process, makes architecture analysis tractable for people who are not security experts. A standard architecture description can be enhanced to provide an explicit picture of information assets that require protection. Standardized icons that are consistently used in UML diagrams, Visio templates, and whiteboard squiggles are especially useful.

[AA2.3] Make SSG available as AA resource or mentor. In order to build an architecture analysis capability outside of the SSG, the SSG advertises itself as a resource or mentor for teams who ask for help conducting their own design review and proactively seek projects to get involved with. The SSG will answer architecture analysis questions during office hours and in some cases might assign someone to sit side-by-side with the architect for the duration of the analysis. In the case of high-risk applications or products, the SSG plays a more active mentorship role.

three

AA Level 3: Build review and remediation capability within the architecture group. Software architects must lead analysis efforts across the organization and must use analysis results to update and create standard architecture patterns that are secure.

[AA3.1] Have software architects lead design review efforts. Software architects throughout the organization lead the architecture analysis process most of the time. The SSG might still contribute to architecture analysis in an advisory capacity or under special circumstances. This activity requires a well understood and well documented architecture analysis process. Even in that case, consistency is very difficult to attain because breaking architectures requires so much experience.

[AA3.2] Drive analysis results into standard architecture patterns. Failures identified during architecture analysis are fed back to the security design committee so that similar mistakes can be prevented in the future through improved design patterns. (See *[SFD3.1 Form a review board or central committee to approve and maintain secure design patterns]*.) Security design patterns can interact in surprising ways that break security. The architecture analysis process should be applied even when vetted design patterns are in standard use.

SSDL TOUCHPOINTS: Code Review (CR)

The overall goal of the Code Review practice is quality control. Those performing code review must ensure the detection and correction of security bugs. The SSG must enforce adherence to standards and the reuse of approved security features.

one

CR Level 1: Manual and automated code review with centralized reporting. The SSG must make itself available to others to raise awareness of and demand for code review. The SSG must perform code reviews on high-risk applications whenever it can get involved in the process and must use the knowledge gained to inform the organization of the types of bugs being discovered. Management must make code review mandatory for all software projects. The SSG must enforce use of centralized tools reporting to capture knowledge on recurring bugs and push that information into strategy and training.

[CR1.1] Create a top N bugs list (real data preferred). The SSG maintains a list of the most important kinds of bugs that need to be eliminated from the organization's code and uses it to drive change. The list helps focus the organization's attention on the bugs that matter most. A generic list could be pulled from public sources, but a list is much more valuable if it is specific to the organization and built from real data gathered from code review, testing, and actual incidents. The SSG can periodically update the list and publish a "most wanted" report. (For another way to use the list, see *[T1.6 Create and use material specific to company history]*.) Some firms use multiple tools and real code base data to build top N lists, not constraining themselves to a particular service or tool. One potential pitfall with a top N list is the problem of "looking for your keys only under the street light." For example, the OWASP Top Ten list rarely reflects an organization's bug priorities. Simply sorting the day's bug data by number of occurrences does not produce a satisfactory Top N list since these data change so often.

[CR1.2] Have SSG perform ad hoc review. The SSG performs an ad hoc code review for high-risk applications in an opportunistic fashion. For example, the SSG might follow up the design review for high-risk applications with a code review. Replace ad hoc targeting with a systematic approach at higher maturity levels. SSG review may involve the use of specific tools and services, or it may be manual.

[CR1.4] Use automated tools along with manual review. Incorporate static analysis into the code review process in order to make code review more efficient and more consistent. The automation does not replace human judgment, but it does bring definition to the review process and security expertise to reviewers who are not security experts. A firm may use an external service vendor as part of a formal code review process for software security. This service should be explicitly connected to a larger SSDL applied during software development and not just "check the security box" on the path to deployment.

[CR1.5] Make code review mandatory for all projects. Code review is a mandatory release gate for all projects under the SSG's purview. Lack of code review or unacceptable results will stop the release train. While all projects must undergo code review, the review process might be different for different kinds of projects. The review for low-risk projects might rely more heavily on automation and the review for high-risk projects might have no upper bound on the amount of time spent by reviewers. In most cases, a code review gate with a minimum acceptable standard forces projects that do not pass to be fixed and re-evaluated before they ship.

[CR1.6] Use centralized reporting to close the knowledge loop and drive training. The bugs found during code review are tracked in a centralized repository. This repository makes it possible to do summary reporting and trend reporting for the organization. The SSG can use the reports to demonstrate progress and drive the training curriculum. (See *[SM2.5 Identify metrics and use them drive budgets]*.) Code review information can be incorporated into a CSO-level dashboard that includes feeds from other parts of the security organization. Likewise, code review information can be fed into a Development-wide project tracking system that rolls up a number of diverse software security feeds (for example: penetration tests, security testing, black box testing, white box testing, etc.). Don't forget that individual bugs make excellent training examples.

two

CR Level 2: Enforce standards through code review process. The SSG must guide developer behavior through coding standards enforcement with automated tools and tool mentors. The SSG must combine automated assessment techniques with tailored rules to find problems efficiently.

[CR2.2] Enforce coding standards. A violation of the organization's secure coding standards is sufficient grounds for rejecting a piece of code. Code review is objective—it does not devolve into a debate about whether or not bad code is exploitable. The enforced portion of the standard could start out being as simple as a list of banned functions. In some cases, coding standards are published as developer guidelines specific to technology stacks (for example, guidelines for C++ or Spring) and then enforced during the code review process or directly in the IDE. Note that guidelines can be positive ("do it this way") as well as negative ("do not use this API").

[CR2.5] Assign tool mentors. Mentors are available to show developers how to get the most out of code review tools. If the SSG is most skilled with the tools, it could use office hours to help developers establish the right configuration or get started interpreting results. Alternatively, someone from the SSG might work with a development team for the duration of the first review they perform. Centralized use of a tool can be distributed into the development organization over time through the use of tool mentors.

[CR2.6] Use automated tools with tailored rules. Customize static analysis to improve efficiency and reduce false positives. Use custom rules to find errors specific to the organization's coding standards or custom middleware. Turn off checks that are not relevant. The same group that provides tool mentoring will likely spearhead the customization. Tailored rules can be explicitly tied to proper usage of technology stacks in a positive sense and avoidance of errors commonly encountered in a firm's code base in a negative sense.

three

CR Level 3: Build an automated code review factory with tailored rules. The SSG must build a capability to find and eradicate specific bugs from the entire codebase.

[CR3.2] Build a factory. Combine assessment results so that multiple analysis techniques feed into one reporting and remediation process. The SSG might write scripts to invoke multiple detection techniques automatically and combine the results into a format that can be consumed by a single downstream review and reporting solution. Analysis engines may combine static and dynamic analysis. The tricky part of this activity is normalizing vulnerability information from disparate sources that use conflicting terminology. In some cases, a CWE-like approach can help with nomenclature. Combining multiple sources helps drive better informed risk mitigation decisions.

[CR3.3] Build a capability for eradicating specific bugs from the entire codebase. When a new kind of bug is found, the SSG writes rules to find it, and uses the rules to identify all occurrences of the new bug throughout the entire codebase. It is possible to entirely eradicate the bug type without waiting for every project to reach the code review portion of its lifecycle. A firm with only a handful of software applications will have an easier time with this activity than firms with a very large number of large apps.

[CR3.4] Automate malicious code detection. Automated code review is used to identify dangerous code written by malicious in-house developers or outsource providers. Examples of malicious code that could be targeted include: backdoors, logic bombs, time bombs, nefarious communication channels, obfuscated program logic, and dynamic code injection. Although out-of-the-box automation might identify some generic malicious-looking constructs, custom rules for static analysis tools used to codify acceptable and unacceptable code patterns in the organization's codebase will quickly become a necessity. Manual code review for malicious code is a good start, but is insufficient to complete this activity.

SSDL TOUCHPOINTS: Security Testing (ST)

The overall goal of the Security Testing practice is quality control performed during the development cycle. Those performing security testing must ensure the detection and correction of security bugs. The SSG must enforce adherence to standards and the reuse of approved security features.

one

ST Level 1: Enhance QA beyond functional perspective. QA must progress to include functional edge and boundary condition testing in its test suites. Test suites must include functional security testing.

[ST1.1] Ensure QA supports edge/boundary value condition testing. The QA team goes beyond functional testing to perform basic adversarial tests. They probe simple edge cases and boundary conditions. No attacker skills required. When QA understands the value of pushing past standard functional testing using acceptable input, they begin to move slowly toward "thinking like a bad guy." A discussion of boundary value testing leads naturally to the notion of an attacker probing the edges on purpose. What happens when you enter the wrong password over and over?

[ST1.3] Drive tests with security requirements and security features. Testers target declarative security mechanisms derived from requirements and security features. For example, a tester could try to access administrative functionality as an unprivileged user or verify that a user account becomes locked after some number of failed authentication attempts. For the most part, security features can be tested in a similar fashion to other software features. Security mechanisms based on requirements such as account lockout, transaction limitations, entitlements, and so on are also tested. Of course, software security is not security software, but getting started with features is easy.

two

ST Level 2: Integrate the attacker perspective into test plans. QA must integrate black-box security testing tools into its process. QA must build test suites for functional security features. The SSG must share its security knowledge and testing results with QA.

[ST2.1] Integrate black box security tools into the QA process. The organization uses one or more black box security testing tools as part of the quality assurance process. The tools are valuable because they encapsulate an attacker's perspective, albeit in a generic fashion. Tools such as IBM Security AppScan or HP WebInspect are relevant for Web applications, and fuzzing frameworks such as Codenomicon are applicable for most network protocols. In some situations, other groups might collaborate with the SSG to apply the tools. For example, a testing team could run the tool, but come to the SSG for help interpreting the results. Regardless of who runs the black box tool, the testing should be properly integrated into the QA cycle of the SSDL.

[ST2.4] Share security results with QA. The SSG routinely shares results from security reviews with the QA department. Over time, QA engineers learn the security mindset. Using security results to inform and evolve particular testing patterns can be a powerful mechanism leading to better security testing. This activity benefits from an engineering-focused QA function that is highly technical.

three

ST Level 3: Deliver risk-based security testing. QA must include security testing in automated regression suites. The SSG must ensure this security testing and its depth is guided by knowledge about the codebase and its associated risks as well as adversarial testing that simulates the attacker's perspective.

[ST3.1] Include security tests in QA automation. Security tests run alongside functional tests as part of automated regression testing. The same automation framework houses both. Security testing is part of the routine. Security tests can be driven from abuse cases identified earlier in the lifecycle or tests derived from creative tweaks of functional tests.

[ST3.2] Perform fuzz testing customized to application APIs. Test automation engineers customize a fuzzing framework to the organization's APIs. They could begin from scratch or use an existing fuzzing toolkit, but customization goes beyond creating custom protocol descriptions or file format templates. The fuzzing framework has a built-in understanding of the application interfaces it calls into. Test harnesses developed explicitly for particular applications can make good places to integrate fuzz testing.

[ST3.3] Drive tests with risk analysis results. Testers use architecture analysis results to direct their work. For example, if the architecture analysis concludes "the security of the system hinges on the transactions being atomic and not being interrupted partway through," then torn transactions will be become a primary target in adversarial testing. Adversarial tests like these can be developed according to risk profile—high-risk flaws first.

[ST3.4] Leverage coverage analysis. Testers measure the code coverage of their security tests in order to identify code that isn't being exercised. Code coverage drives increased security testing depth. Standard-issue black box testing tools achieve exceptionally low coverage, leaving a majority of the software under test unexplored. Don't let this happen to your tests. Using standard measurements for coverage such as function coverage, line coverage or multiple condition coverage is fine.

[ST3.5] Begin to build and apply adversarial security tests (abuse cases). Testing begins to incorporate test cases based on abuse cases provided by the SSG. Testers move beyond verifying functionality and take on the attacker's perspective. For example, testers might systematically attempt to replicate incidents from the organization's history. Abuse and misuse cases based on the attacker's perspective can also be driven from security policies, attack intelligence, and guidelines. This turns the corner from testing features to attempting to break the software under test.

DEPLOYMENT: Penetration Testing (PT)

The overall goal of the Penetration Testing practice is quality control of software that has moved into deployment. Those performing penetration testing must ensure the detection and correction of security defects. The SSG must enforce adherence to standards and the reuse of approved security features.

one

PT Level 1: Remediate penetration testing results. Managers and the SSG must initiate the penetration testing process, with internal or external resources. Managers and the SSG must ensure that deficiencies discovered are addressed and that everyone is made of aware of progress.

[PT1.1] Use external penetration testers to find problems. Many organizations are not willing to address software security until there is unmistakable evidence that the organization is not somehow magically immune to the problem. If security has not been a priority, external penetration testers demonstrate that the organization's code needs help. Penetration testers could be brought in to break a high-profile application in order to make the point. Over time, the focus of penetration testing moves from "I told you our stuff was broken" to a smoke test and sanity check done before shipping. External penetration testers bring a new set of eyes to the problem.

[PT1.2] Feed results to the defect management and mitigation system. Penetration testing results are fed back to development through established defect management or mitigation channels, and development responds using their defect management and release process. The exercise demonstrates the organization's ability to improve the state of security. Many firms are beginning to emphasize the critical importance of not just identifying but more importantly fixing security problems. One way to ensure attention is to add a security flag to the bug tracking and defect management system. Evolving DevOps and integrated team structures do not eliminate the need for formalized defect management systems.

[PT1.3] Use penetration testing tools internally. The organization creates an internal penetration testing capability that makes use of tools. This capability can be part of the SSG or part of a specialized and trained team elsewhere in the organization. The tools improve efficiency and repeatability of the testing process. Tools can include off the shelf products, standard issue network penetration tools that understand the application layer, and hand-written scripts.

two

PT Level 2: Schedule regular penetration testing by informed, internal penetration testers. The SSG must create an internal penetration testing capability that is periodically applied to all applications. The SSG must share its security knowledge and testing results with all penetration testers.

[PT2.2] Provide penetration testers with all available information. Penetration testers, whether internal or external, use all available information about their target. Penetration testers can do deeper analysis and find more interesting problems when they have source code, design documents, architecture analysis results, and code review results. Give penetration testers everything you have created throughout the SSDL. If your penetration tester doesn't ask for the code, you need a new penetration tester.

[PT2.3] Schedule periodic penetration tests for application coverage. Periodically test all applications in the SSG's purview according to an established schedule (which could be tied to the calendar or to the release cycle). The testing serves as a sanity check and helps ensure yesterday's software isn't vulnerable to today's attacks. High-profile applications might get a penetration test at least once a year. One important aspect of periodic testing is to make sure that the problems identified in a penetration test are actually fixed and they don't creep back into the build.

three

PT Level 3: Carry out deep-dive penetration testing. Managers must ensure that the organization's penetration testing knowledge keeps pace with advances by attackers. The SSG must take advantage of organizational knowledge to customize penetration testing tools.

[PT3.1] Use external penetration testers to perform deep-dive analysis. The organization uses external penetration testers to do deep-dive analysis for critical projects and to introduce fresh thinking into the SSG. These testers are experts and specialists. They keep the organization up to speed with the latest version of the attacker's perspective and they have a track record for breaking the type of software being tested. Skilled penetration testers will always break a system. The question is whether they demonstrate new kinds of thinking about attacks that can be useful when designing, implementing, and hardening new systems. Creating new types of attacks from threat intelligence and abuse cases prevents checklist-driven approaches that only look for known types of problems.

[PT3.2] Have the SSG customize penetration testing tools and scripts. The SSG either creates penetration testing tools or adapts publicly available tools so they can more efficiently and comprehensively attack the organization's systems. Tools improve the efficiency of the penetration testing process without sacrificing the depth of problems the SSG can identify. Tools that can be tailored are always preferable to generic tools.

DEPLOYMENT: Software Environment (SE)

The overall goal of the Software Environment practice is change management. Those responsible for the software environment must ensure their ability to make authorized changes and to detect unauthorized changes and activity. Managers must enforce adherence to corporate policy.

one

SE Level 1: Ensure the application environment supports software security. The operations group ensures required host and network security controls are functioning and proactively monitors software, including application inputs.

[SE1.1] Use application input monitoring. The organization monitors the input to software it runs in order to spot attacks. For web code, a web application firewall (WAF) can do the job. The SSG could be responsible for the care and feeding of the system. Responding to attack is not part of this activity. Defanged WAFs that write log files can be useful if somebody reviews the logs periodically. On the other hand, a WAF that is unmonitored makes no noise when an application falls in the woods.

[SE1.2] Ensure host and network security basics are in place. The organization provides a solid foundation for software by ensuring that host and network security basics are in place. It is common for operations security teams to be responsible for duties such as patching operating systems and maintaining firewalls. Doing software security before network security is like putting on your pants before putting on your underwear.

two

SE Level 2: Use published installation guides and code signing. The SSG must ensure software development processes protect code integrity. The SSG must ensure that application installation and maintenance guides are created for the operations group to use.

[SE2.2] Publish installation guides. The SSDL requires the creation of an installation guide to help operators install and configure the software securely. If special steps are required in order to ensure a deployment is secure, the steps are outlined in the installation guide. The guide should include discussion of COTS components. In some cases, installation guides are distributed to customers who buy the software. Evolving DevOps and integrated team structures do not eliminate the need for written guides. Of course, secure by default is always the best way to go.

[SE2.4] Use code signing. The organization uses code signing for software published across trust boundaries. Code signing is particularly useful for protecting the integrity of software that leaves the organization's control, such as shrink-wrapped applications or thick clients. The fact that some mobile platforms require application code to be signed does not indicate institutional use of code signing.

three

SE Level 3: Protect client-side code and actively monitor software behavior. The SSG must ensure that all code leaving the organization is protected. The operations group must monitor software behavior.

[SE3.2] Use code protection. In order to protect intellectual property and make exploit development harder, the organization erects barriers to reverse engineering. Obfuscation techniques could be applied as part of the production build and release process. Employing platform-specific controls such as Data Execution Prevention (DEP), Safe Structured Error Handling (SafeSEH), and Address Space Layout Randomization (ASLR) can make exploit development more difficult.

[SE3.3] Use application behavior monitoring and diagnostics. The organization monitors the behavior of production software looking for misbehavior and signs of attack. This activity goes beyond host and network monitoring to look for problems that are specific to the software, such as indications of fraud. Intrusion detection and anomaly detection systems at the application level may focus on an application's interaction with the operating system (through system calls) or with the kinds of data that an application consumes, originates, and manipulates.

(blank page)

DEPLOYMENT: Configuration Management and Vulnerability Management (CMVM)

The overall goal of the Configuration Management and Vulnerability Management practice is change management. The SSG and application owners must ensure their ability to track authorized changes to applications and to detect unauthorized changes and activity. Application owners must enforce adherence to corporate policy.

one

CMVM Level 1: Use operations monitoring data to drive developer behavior. The SSG must support incident response. The SSG must use operations data to suggest changes in the SSDL and developer behavior.

[CMVM1.1] Create or interface with incident response. The SSG is prepared to respond to an incident. The group either creates its own incident response capability or interfaces with the organization's existing incident response team. A regular meeting between the SSG and the incident response team can keep information flowing in both directions. In many cases, software security initiatives have evolved from incident response teams who began to realize that software vulnerabilities were the bane of their existence.

[CMVM1.2] Identify software defects found in operations monitoring and feed them back to development. Defects identified through operations monitoring are fed back to development and used to change developer behavior. The contents of production logs can be revealing (or can reveal the need for improved logging). In some cases, providing a way to enter incident triage data into an existing bug tracking system (many times making use of a special security flag) seems to work. The idea is to close the information loop and make sure security problems get fixed. In the best of cases, processes in the SSDL can be improved based on operational data.

two

CMVM Level 2: Ensure that emergency response is available during application attack. Managers and the SSG must support emergency response to ongoing application attacks. Managers and the SSG must maintain a code inventory. The SSG uses operations data to direct evolution in the SSDL and in developer behavior.

[CMVM2.1] Have emergency codebase response. The organization can make quick code changes when an application is under attack. A rapid-response team works in conjunction with the application owners and the SSG to study the code and the attack, find a resolution, and push a patch into production. Often times, the emergency response team is the development team itself. Fire drills do not count; a well-defined process is required.

[CMVM2.2] Track software bugs found in operations through the fix process. Defects found in operations are fed back to development, entered into established defect management systems, and tracked through the fix process. This capability could come in the form of a two-way bridge between the bug finders and the bug fixers. Make sure the loop is closed completely. Setting a security flag in the bug tracking system can help facilitate tracking.

[CMVM2.3] Develop an operations inventory of applications. The organization has a map of its software deployments. If a piece of code needs to be changed, operations can reliably identify all of the places where the change needs to be installed. Sometimes common components shared between multiple projects are noted so that when an error occurs in one application, other applications that share the same components can be fixed as well.

three

CMVM Level 3: Create a tight loop between operations and development. The SSG must ensure the SSDL both addresses code deficiencies found in operations and includes enhancements that eliminate associated root causes.

[CMVM3.1] Fix all occurrences of software bugs found in operations. The organization fixes all instances of each software bug found during operations and not just the small number of instances that have triggered bug reports. This requires the ability to reexamine the entire codebase when new kinds of bugs come to light. (See *[CR3.3] Build capability for eradicating specific bugs from entire codebase*.) One way to approach this is to create a rule set that generalizes a deployed bug into something that can be scanned for using automated code review.

[CMVM3.2] Enhance the SSDL to prevent software bugs found in operations. Experience from operations leads to changes in the SSDL. The SSDL is strengthened to prevent the reintroduction of bugs found during operations. To make this process systematic, the incident response post mortem could include a "feedback to SSDL" step. This works best when root cause analysis pinpoints where in the SDLC an error may have been introduced or slipped by uncaught. An ad hoc approach is not sufficient.

[CMVM3.3] Simulate software crisis. The SSG simulates high-impact software security crises to ensure software incident response capabilities minimize damage. Simulations could test for the ability to identify and mitigate specific threats or, in other cases, could begin with the assumption that a critical system or service is already compromised and evaluate the organization's ability to respond. When simulations model successful attacks, an important question to consider is the time period required to clean things up. Regardless, simulations must focus on security-relevant software failure and not natural disasters or other types of emergency response drills. If the data center is burning to the ground, the SSG won't be among the first responders.

[CMVM3.4] Operate a bug bounty program. The organization solicits vulnerability reports from external researchers and pays a bounty for each verified and accepted vulnerability received. Payouts typically follow a sliding scale linked to multiple factors, such as vulnerability type (e.g., remote code execution is worth $10,000 versus CSRF is worth $750), exploitability (demonstrable exploits command much higher payouts), or specific services and software versions (widely-deployed or critical services warrant higher payouts). Ad hoc or short-duration activities, such as capture-the-flag contests, do not count. [This is a new activity that will be reported on in BSIMM6.]

The BSIMM Skeleton

The BSIMM skeleton provides a way to view the maturity model at a glance and is useful when assessing a software security program. The skeleton includes one page per practice organized by three levels. Each activity is associated with an objective. More complete descriptions of the activities, examples, and term definition can be found in the main document

GOVERNANCE: STRATEGY AND METRICS Planning, assigning roles and responsibilities, identifying software security goals, determining budgets, identifying metrics and gates.		
Objective	**Activity**	**Level**
[SM1.1] make the plan explicit	publish process (roles, responsibilities, plan), evolve as necessary	1
[SM1.2] build support throughout organization	create evangelism role and perform internal marketing	
[SM1.3] secure executive buy-in	educate executives	
[SM1.4] establish SSDL gates (but do not enforce)	identify gate locations, gather necessary artifacts	
[SM1.6] make clear who's taking the risk	require security sign-off	
[SM2.1] foster transparency (or competition)	publish data about software security internally	2
[SM2.2] change behavior	enforce gates with measurements and track exceptions	
[SM2.3] create broad base of support	create or grow a satellite	
[SM2.5] define success	identify metrics and use them to drive budgets	
[SM3.1] know where all apps in your inventory stand	use an internal tracking application with portfolio view	3
[SM3.2] create external support	run an external marketing program	

GOVERNANCE: COMPLIANCE AND POLICY

Identifying controls for compliance regimens, developing contractual controls (COTS SLA), setting organizational policy, auditing against policy.

	Objective	Activity	Level
[CP1.1]	understand compliance drivers (FFIEC, GLBA, OCC, PCI, SOX, SAS 70, HIPAA)	unify regulatory pressures	1
[CP1.2]	promote privacy	identify PII obligations	
[CP1.3]	meet regulatory needs or customer demand with a unified approach	create policy	
[CP2.1]	promote privacy	identify PII data inventory	2
[CP2.2]	ensure accountability for software risk	require security sign-off for compliance-related risk	
[CP2.3]	align practices with compliance	implement and track controls for compliance	
[CP2.4]	ensure vendors don't screw up compliance	paper all vendor contracts with software security SLAs	
[CP2.5]	gain executive buy-in	promote executive awareness of compliance and privacy obligations	
[CP3.1]	demonstrate compliance story	create regulator eye-candy	3
[CP3.2]	manage third-party vendors	impose policy on vendors	
[CP3.3]	keep policy aligned with reality	drive feedback from SSDL data back to policy (T: strategy/metrics)	

	GOVERNANCE: TRAINING		
	Objective	**Activity**	**Level**
[T1.1]	promote culture of security throughout the organization	provide awareness training	1
[T1.5]	build capabilities beyond awareness	deliver role-specific advanced curriculum (tools, technology stacks, bug parade)	
[T1.6]	see yourself in the problem	create and use material specific to company history	
[T1.7]	reduce impact on training targets and build delivery staff	deliver on-demand individual training	
[T2.5]	educate/strengthen social network	enhance satellite through training and events	2
[T2.6]	ensure new hires enhance culture	include security resources in onboarding	
[T2.7]	create social network tied into dev	identify satellite through training	
[T3.1]	align security culture with career path	reward progression through curriculum (certification or HR)	3
[T3.2]	spread security culture to providers	provide training for vendors or outsourced workers	
[T3.3]	market security culture as differentiator	host external software security events	
[T3.4]	keep staff up-to-date and address turnover	require an annual refresher	
[T3.5]	act as informal resource to leverage teachable moments	establish SSG office hours	

INTELLIGENCE: ATTACK MODELS

Threat modeling, abuse cases, data classification, technology-specific attack patterns.

	Objective	Activity	Level
[AM1.1]	understand attack basics	build and maintain a top N possible attacks list	1
[AM1.2]	prioritize applications by data consumed/manipulated	create a data classification scheme and inventory	
[AM1.3]	understand the "who" of attacks	identify potential attackers	
[AM1.4]	understand the organization's history	collect and publish attack stories	
[AM1.5]	stay current on attack/vulnerability environment	gather attack intelligence	
[AM1.6]	communicate attacker perspective	build an internal forum to discuss attacks (T: standards/req)	
[AM2.1]	provide resources for security testing and AA	build attack patterns and abuse cases tied to potential attackers	2
[AM2.2]	understand technology-driven attacks	create technology-specific attack patterns	
[AM3.1]	get ahead of the attack curve	have a science team that develops new attack methods	3
[AM3.2]	arm testers and auditors	create and use automation to do what the attackers will do	

INTELLIGENCE: SECURITY FEATURES AND DESIGN

Security patterns for major security controls, middleware frameworks for controls, proactive security guidance.

	Objective	Activity	Level
[SFD1.1]	create proactive security guidance around security features	build and publish security features	1
[SFD1.2]	inject security thinking into architecture group	engage SSG with architecture	
[SFD2.1]	create proactive security design based on technology stacks	build secure-by-design middleware frameworks and common libraries (T: code review)	2
[SFD2.2]	address the need for new architecture	create SSG capability to solve difficult design problems	
[SFD3.1]	formalize consensus on design	form a review board or central committee to approve and maintain secure design patterns	3
[SFD3.2]	promote design efficiency	require use of approved security features and frameworks (T: AA)	
[SFD3.3]	practice reuse	find and publish mature design patterns from the organization	

INTELLIGENCE: STANDARDS AND REQUIREMENTS

Explicit security requirements, recommended COTS, standards for major security controls, standards for technologies in use, standards review board.

	Objective	Activity	Level
[SR1.1]	meet demand for security features	create security standards (T: sec features/design)	1
[SR1.2]	ensure that everybody knows where to get latest and greatest	create a security portal	
[SR1.3]	compliance strategy	translate compliance constraints to requirements	
[SR1.4]	tell people what to look for in code review	use secure coding standards	
[SR2.2]	formalize standards process	create a standards review board	2
[SR2.3]	reduce SSG workload	create standards for technology stacks	
[SR2.4]	manage open source risk	identify open source	
[SR2.5]	gain buy-in from legal department and standardize approach	create SLA boilerplate (T: compliance and policy)	
[SR3.1]	manage open source risk	control open source risk	3
[SR3.2]	educate third-party vendors	communicate standards to vendors	

SSDL TOUCHPOINTS: ARCHITECTURE ANALYSIS

Capturing software architecture diagrams, applying lists of risks and threats, adopting a process for review, building an assessment and remediation plan.

	Objective	Activity	Level
[AA1.1]	get started with AA	perform security feature review	1
[AA1.2]	demonstrate value of AA with real data	perform design review for high-risk applications	
[AA1.3]	build internal capability on security architecture	have SSG lead design review efforts	
[AA1.4]	have a lightweight approach to risk classification and prioritization	use a risk questionnaire to rank applications	
[AA2.1]	model objects	define and use AA process	2
[AA2.2]	promote a common language for describing architecture	standardize architectural descriptions (including data flow)	
[AA2.3]	build capability organization-wide	make SSG available as AA resource or mentor	
[AA3.1]	build capabilities organization-wide	have software architects lead design review efforts	3
[AA3.2]	build proactive security architecture	drive analysis results into standard architecture patterns (T: sec features/design)	

SSDL TOUCHPOINTS: CODE REVIEW

Use of code review tools, development of customized rules, profiles for tool use by different roles, manual analysis, ranking/measuring results.

	Objective	Activity	Level
[CR1.1]	know which bugs matter to you	create top N bugs list (real data preferred) (T: training)	1
[CR1.2]	review high-risk applications opportunistically	have SSG perform ad hoc review	
[CR1.4]	drive efficiency/consistency with automation	use automated tools along with manual review	
[CR1.5]	find bugs earlier	make code review mandatory for all projects	
[CR1.6]	know which bugs matter (for training)	use centralized reporting to close the knowledge loop and drive training (T: strategy/metrics)	
[CR2.2]	drive behavior objectively	enforce coding standards	2
[CR2.5]	make most efficient use of tools	assign tool mentors	
[CR2.6]	drive efficiency/reduce false positives	use automated tools with tailored rules	
[CR3.2]	combine assessment techniques	build a factory	3
[CR3.3]	handle new bug classes in an already scanned codebase	build capability for eradicating specific bugs from the entire codebase	
[CR3.4]	address insider threat from development	automate malicious code detection	

SSDL TOUCHPOINTS: SECURITY TESTING

Use of black box security tools in QA, risk driven white box testing, application of the attack model, code coverage analysis.

	Objective	Activity	Level
[ST1.1]	execute adversarial tests beyond functional	ensure QA supports edge/boundary value condition testing	1
[ST1.3]	start security testing in familiar functional territory	drive tests with security requirements and security features	
[ST2.1]	use encapsulated attacker perspective	integrate black box security tools into the QA process	2
[ST2.4]	facilitate security mindset	share security results with QA	
[ST3.1]	include security testing in regression	include security tests in QA automation	3
[ST3.2]	teach tools about your code	perform fuzz testing customized to application APIs	
[ST3.3]	probe risk claims directly	drive tests with risk analysis results	
[ST3.4]	drive testing depth	leverage coverage analysis	
[ST3.5]	move beyond functional testing to attacker's perspective	begin to build and apply adversarial security tests (abuse cases)	

DEPLOYMENT: PENETRATION TESTING

Vulnerabilities in final configuration, feeds to defect management and mitigation.

	Objective	Activity	Level
[PT1.1]	demonstrate that your organization's code needs help too	use external penetration testers to find problems	1
[PT1.2]	fix what you find to show real progress	feed results to the defect management and mitigation system (T: config/vuln mgmt)	
[PT1.3]	create internal capability	use penetration testing tools internally	
[PT2.2]	promote deeper analysis	provide penentration testers with all available information (T: AA & code review)	2
[PT2.3]	sanity check constantly	schedule periodic penetration tests for application coverage	
[PT3.1]	keep up with edge of attacker's perspective	use external penetration testers to perform deep-dive analysis	3
[PT3.2]	automate for efficiency without losing depth	have the SSG customize penetration testing tools and scripts	

DEPLOYMENT: SOFTWARE ENVIRONMENT

OS and platform patching, Web application firewalls, installation and configuration documentation, application monitoring, change management, code signing.

	Objective	Activity	Level
[SE1.1]	watch software	use application input monitoring	1
[SE1.2]	provide a solid host/network foundation for software	ensure host and network security basics are in place	
[SE2.2]	guide operations on application needs	publish installation guides	2
[SE2.4]	protect apps (or parts of apps) that are published over trust boundaries	use code signing	
[SE3.2]	protect IP and make exploit development harder	use code protection	3
[SE3.3]	watch software	use application behavior monitoring and diagnostics	

DEPLOYMENT: CONFIGURATION MANAGEMENT AND VULNERABILITY MANAGEMENT

Patching and updating applications, version control, defect tracking and remediation, incident handling.

	Objective	Activity	Level
[CMVM1.1]	know what to do when something bad happens	create or interface with incident response	1
[CMVM1.2]	use ops data to change dev behavior	identify software defects found in operations monitoring and feed them back to development	
[CMVM2.1]	be able to fix apps when they are under direct attack	have emergency codebase response	2
[CMVM2.2]	use ops data to change dev behavior	track software bugs found in operations through the fix process	
[CMVM2.3]	know where the code is	develop an operations inventory of applications	
[CMVM3.1]	learn from operational experience	fix all occurrences of software bugs found in operations (T: code review)	3
[CMVM3.2]	use ops data to change dev behavior	enhance the SSDL to prevent software bugs found in operations	
[CMVM3.3]	ensure processes are in place to minimize software incident impact	simulate software crisis	
[CMVM3.4]	*engage external researchers in vulnerability discovery*	*operate a bug bounty program*	

Ranking BSIMM-V Activities

Choosing which of the BSIMM activities to adopt and in what order can be a challenge. We suggest creating a software security initiative strategy and plan by focusing on goals and objectives first and letting the activities select themselves. Creating a timeline for rollout is often very useful.

Of course, learning from experience is also a good strategy. Toward that end, this section is devoted to describing the set of core activities that we observed in at least forty-three of the sixty-seven organizations we studied and then providing a chart of all 112 activities with a summary score that can be seen as a rough weighting.

Core BSIMM Activities

Of the 112 activities observed in BSIMM-V, there are twelve activities that at least forty-three of the sixty-seven firms we studied carry out (64%), one identified in each practice. Though we can't directly conclude that these twelve activities are necessary for all software security initiatives, we can say with confidence that these activities are commonly found in highly successful programs. This suggests that if you are working on an initiative of your own, you should consider these twelve activities particularly carefully (not to mention the other 100).

Twelve Core Activities Everybody Does		
	Objective	Activity
[SM1.4]	establish SSDL gates (but do not enforce)	identify gate locations, gather necessary artifacts
[CP1.2]	promote privacy	identify PII obligations
[T1.1]	promote culture of security throughout the organization	provide awareness training
[AM1.2]	prioritize applications by data consumed/manipulated	create a data classification scheme and inventory
[SFD1.1]	create proactive security guidance around security features	build and publish security features
[SR1.1]	meet demand for security features	create security standards
[AA1.1]	get started with AA	perform security feature review
[CR1.4]	drive efficiency/consistency with automation	use automated tools along with manual review
[ST1.3]	start security testing in familiar functional territory	drive tests with security requirements and security features
[PT1.1]	demonstrate that your organization's code needs help too	use external penetration testers to find problems
[SE1.2]	provide a solid host/network foundation for software	ensure host and network security basics are in place
[CMVM1.2]	use ops data to change dev behavior	identify software bugs found in operations monitoring and feed them back to development

Activities Observed over Sixty-Seven Firms

The chart on the next page shows how many of the sixty-seven firms we studied have adopted various activities. The twelve core activities are highlighted in yellow. Though you can use this as a rough "weighting" of activities by prevalence, a software security initiative plan is best approached through goals and objectives.

Governance		Intelligence		SSDL Touchpoints		Deployment	
Activity	Observed	Activity	Observed	Activity	Observed	Activity	Observed
[SM1.1]	44	[AM1.1]	21	[AA1.1]	56	[PT1.1]	62
[SM1.2]	34	[AM1.2]	43	[AA1.2]	35	[PT1.2]	51
[SM1.3]	34	[AM1.3]	30	[AA1.3]	24	[PT1.3]	43
[SM1.4]	57	[AM1.4]	12	[AA1.4]	42	[PT2.2]	24
[SM1.6]	36	[AM1.5]	42	[AA2.1]	10	[PT2.3]	27
[SM2.1]	26	[AM1.6]	16	[AA2.2]	8	[PT3.1]	13
[SM2.2]	31	[AM2.1]	7	[AA2.3]	20	[PT3.2]	8
[SM2.3]	27	[AM2.2]	11	[AA3.1]	11		
[SM2.5]	20	[AM3.1]	4	[AA3.2]	4		
[SM3.1]	16	[AM3.2]	6				
[SM3.2]	6						
[CP1.1]	43	[SFD1.1]	54	[CR1.1]	24	[SE1.1]	34
[CP1.2]	52	[SFD1.2]	53	[CR1.2]	34	[SE1.2]	61
[CP1.3]	45	[SFD2.1]	26	[CR1.4]	50	[SE2.2]	31
[CP2.1]	24	[SFD2.2]	29	[CR1.5]	23	[SE2.4]	25
[CP2.2]	28	[SFD2.3]	9	[CR1.6]	25	[SE3.2]	10
[CP2.3]	29	[SFD3.1]	13	[CR2.2]	10	[SE3.3]	9
[CP2.4]	25	[SFD3.2]	9	[CR2.5]	15		
[CP2.5]	35			[CR3.1]	18		
[CP3.1]	14			[CR3.2]	4		
[CP3.2]	11			[CR3.3]	6		
[CP3.3]	8			[CR3.4]	1		
[T1.1]	50	[SR1.1]	48	[ST1.1]	51	[CMVM1.1]	59
[T1.5]	29	[SR1.2]	43	[ST1.3]	55	[CMVM1.2]	59
[T1.6]	23	[SR1.3]	45	[ST2.1]	27	[CMVM2.1]	50
[T1.7]	33	[SR1.4]	27	[ST2.3]	13	[CMVM2.2]	44
[T2.5]	9	[SR2.1]	23	[ST2.4]	11	[CMVM2.3]	30
[T2.6]	13	[SR2.2]	19	[ST3.1]	8	[CMVM3.1]	6
[T2.7]	9	[SR2.3]	19	[ST3.2]	6	[CMVM3.2]	6
[T3.1]	4	[SR2.4]	22	[ST3.3]	5	[CMVM3.3]	2
[T3.2]	4	[SR2.5]	8	[ST3.4]	7		
[T3.3]	8	[SR3.1]	12				
[T3.4]	9						
[T3.5]	5						

Appendix: Adjusting BSIMM4 for BSIMM-V

Because the BSIMM is a data driven model, we have chosen to make adjustments to the model based on the data observed between BSIMM4 and BSIMM-V.

In order to preserve backwards compatibility, all changes have been made by adding new activity labels to the model, even when an activity has simply been moved between levels.

We made all changes by considering outliers both in the model itself, and in the levels we assigned to various activities in the twelve practices. We used the results of an intra-level standard deviation analysis to determine which "outlier" activities to move between levels. To do this, we focused on changes that minimize standard deviation in the average number of observed activities at each level. Our hard and fast rule was to ensure that on average the number of observed activities per level followed a logical progression from common to rare (as outlined in our discussion of levels).

Here are the four changes we made according to that paradigm:

1. [SFD2.3] became [SFD3.3], [SFD2.3] was removed (level promotion)
2. [SR2.1] became [SR3.2], [SR2.1] was removed (level promotion)
3. [ST2.3] became [ST3.5], [ST2.3] was removed (level promotion)
4. [CR3.1] became [CR2.6], [CR3.1] was removed (level demotion)

We also carefully considered, but did not adjust, the following activities: [CP2.5], [T3.3], [T3.4], [AM1.4]

The resulting BSIMM-V scorecard can be seen on page 60.